

Appendix A

Event Generator for Inclusive $b \rightarrow u \ell \nu$

We describe the InclGen event generator, developed to simulate $b \rightarrow u \ell \nu$ decays using a hybrid approach that combines the detailed expectations derived from models of the low-lying hadronic resonances with constraints from the broader inclusive theory.

A.1 Introduction

The weak decay $b \rightarrow u \ell \nu$ offers experimental access to the important CKM element $|V_{ub}|$, but extraction of this parameter is made difficult by the challenge of making theoretical calculations in the restricted regions of phase space where the tiny signal is not dwarfed by the overwhelming background from kinematically similar $b \rightarrow c \ell \nu$ decays. The small windows where observation of $b \rightarrow u \ell \nu$ is experimentally feasible are often the same areas where traditional inclusive calculations begin to break down.

The challenge of simulating $b \rightarrow u \ell \nu$ is complicated by the limited experimental knowledge we have of the decay. The $b \rightarrow u \ell \nu$ hadronic mass spectrum includes at least a dozen different exclusive resonances, whose total does not saturate the total inclusive rate, implying that there is also a significant non-resonant component about which very little is known.

As part of a new inclusive $b \rightarrow u \ell \nu$ analysis at CLEO, we have developed an event generator that attempts to make inroads on these problems by combining information from both the inclusive theory and the exclusive models in a self-consistent way to gain as much predictive power—and as little explicit model dependence—as possible. The resulting generator, called InclGen, generates B decays along the weak channel $B \rightarrow X_u \ell \nu$. The resulting hadronic system X_u formed from the u quark and other light degrees of freedom present in the initial B meson is one of several ISGW2 resonances or a non-resonant, multi-particle system. The generator uses the hadronic mass spectrum from the inclusive theory to dictate the mass distribution of the non-resonant rate as well as the relative ratio of resonant to non-resonant decays.

Currently the ACCMM [96] and HQET [98] inclusive calculations are supported, and the masses and widths of the various exclusive resonances can be adjusted at the user's discretion. Additional parameters offer the ability to tune exactly how the inclusive rate is divided among the exclusive modes, and so how the resulting non-resonant rate is distributed.

The InclGen generator is designed to run as part of CLEO’s standard event generation framework (called QQ), although it can be run independently in a less flexible, stand-alone mode. Rather than touring each of 20,000 lines of code, here we attempt to simply give the flavor of what the package does, what the standard behaviors are, and how to modify some of the simpler features.

As an aside, we note that each of the B factories, and indeed some of the past LEP experiments, have developed their own “hybrid” event generators for $b \rightarrow u \ell \nu$, all of which make some attempt to blend the reality of low-lying hadronic resonances with the mass spectrum determined from the broader inclusive theory. See Ref [126], for instance, for a discussion of the simulation package in use at BaBar.

A.2 Theory Inputs

The InclGen simulation relies on a theoretical prediction of the triply-differential inclusive spectrum $d^3\Gamma/dE_\ell dM_X^2 dq^2$ and on phenomenological models for the various exclusive modes. We discuss each of these inputs in more detail below.

A.2.1 Inclusive Theory

An HQET-based description of the inclusive decay is the default for the simulation, although a description based on the older ACCMM model [96] is also supported. The HQET prediction is from a 1999 paper by DeFazio and Neubert [98] that computes the triply-differential rate to leading-order in $1/M_B$ and α_s . The final physical rate is obtained from the perturbative calculation by convolving with the non-perturbative shape function, taken to be of the exponential form (also referred to in the same paper):

$$f(k_+) = N(1 - x)^a e^{(1+a)x}. \quad (\text{A.1})$$

Here, $x \equiv \frac{k_+}{\Lambda}$ is a reduced light-cone momentum variable; the parameter a is adjusted to match the few known constraints on the moments of the shape function; and the factor N allows for unit normalization. The entire calculation has been coded and tested at CLEO as a stand-alone C++ routine, described in Ref [99]. Recently, the code has been extended to include support for the so-called “Roman” and “Gaussian” forms for the shape function, but they are not used in the standard InclGen simulation.¹

¹At present, (only) a simple modification of the InclGen code is necessary to choose a different shape function; recompilation is still required. Although the choice is not (yet) a run-time option, it could easily be made into one.

As part of the recent CLEO lepton endpoint analyses [128], the shape function parameters have been estimated from the photon energy spectrum observed in $b \rightarrow s\gamma$ [100, 127]. We use the results of those studies as the nominal values for the inclusive calculation carried out in InclGen. In particular, we take

$$\overline{\Lambda}^{\text{SF}} = 0.545 \text{ GeV} \quad (\text{A.2})$$

$$\lambda_1^{\text{SF}} = -0.432 \text{ GeV}^2 \quad (\text{A.3})$$

$$\alpha_s = 0.22 \quad (\text{A.4})$$

$$M_B = 5.28 \text{ GeV}. \quad (\text{A.5})$$

Doubly and singly differential distributions are easily obtained from the full expression for $d^3\Gamma$ by using Monte Carlo integration to eliminate the irrelevant variables.

A.2.2 Exclusive Resonances

Many of the charmless hadronic resonances X_u predicted by the ISGW2 quark model [117] have been unambiguously identified, and we use the complete set of ISGW2 states as a reasonable proxy for the true set of resonant modes found in the $b \rightarrow u\ell\nu$ spectrum. Further, when a resonant channel is selected (according to the method described in the next section), the kinematics of the three-body final state $X_u\ell\nu$ are generated according to the ISGW2 description of the relevant form factors. In this way, all exclusive decays are simulated in the context of the same model. Although more recent and less model-dependent calculations exist for some of the pseudoscalar and vector modes, we treat these improvements as sub-leading corrections to the broad features of the hadronic mass spectrum. (This assumption also significantly simplifies the coding challenge.) We note that since the exclusive decays are actually executed by the EvtGen [86] package, different exclusive models can be readily accommodated by making simple changes in the EvtGen decay files. Consult the EvtGen documentation for more information on making such changes in the models used for exclusive semileptonic decays.

We emphasize that, in particular, we rely on the ISGW2 predictions for the *relative* partial widths (branching fractions) of the various exclusive modes.² In addition, we compare the total rate predicted by the inclusive theory (modulo the unknown but common factor of $|V_{ub}|^2$) directly to the partial rate expected for each exclusive channel. This comparison in effect directly constrains the total magnitude of the non-resonant rate. The total $b \rightarrow u\ell\nu$ rate, including resonant and non-resonant contributions, is simply controlled by adjusting the inclusive $b \rightarrow u\ell\nu$ branching fraction assigned to InclGen in QQ's `decay.dec` file. For more details, see Sec A.5.

²That is, we do not use experimental determinations of the various branching fractions; we use the numbers in the ISGW2 paper.

The $a_0(1450)$

The $a_0(980)$ resonance has occasionally been identified with the 1^3P_0 resonance of the ISGW2 model. We instead map the 1^3P_0 onto the $a_0(1450)$, a resonance previously not included in the default QQ simulation. The InclGen decay files thus include the introduction of this new particle, charged and neutral, with a mass of 1450 MeV and decay modes based on the observations reported in the 2000 Particle Data Book [5].

A.3 Algorithm

The inclusive hadronic mass spectrum is the key component in the simulation, since it is used to determine how the total inclusive rate is to be divided between the exclusive resonances and a remaining catch-all non-resonant component. Essentially, a piece of the inclusive spectrum is carved out for each exclusive mode. The intuition is that since the inclusive spectrum has averaged over hadronic details finer than the hadronic binding scale Λ_{QCD} , each exclusive mode can be associated a (overlapping) portion of the inclusive spectrum with size scale $\mathcal{O}(\Lambda_{\text{QCD}})$.

To implement this scheme in practice, each ISGW2 mode is assigned a “weight function” that has a width characterized by a user parameter Λ , expected to loosely correspond to Λ_{QCD} . Each weight function represents the portion of the inclusive mass spectrum that is “borrowed” or reserved for that particular exclusive mode for the purposes of the simulation. The sum of all of the weight functions thus determines, by implicit subtraction from the inclusive spectrum, the shape of the non-resonant component. Note that these weight functions do not represent the true mass distributions of the resonances, but only the region of the inclusive spectrum over which they “suck up” inclusive rate.

In order to accommodate the clustering of X_u resonances in the low-mass region near $M_X \sim 1$ GeV, several different weight functions are supported, allowing the user fine control over what parts of the inclusive spectrum are assigned to a particular resonant mode. The normalization of each weight function is naturally forced to match the partial width of the particular exclusive mode.³ In this way, the exclusive partial widths are indirectly compared to the total available inclusive rate, as well as to the inclusive rate available at each interval dM_X . For flexibility, we support the following weight functions, all centered on the resonance’s nominal mass and with independent widths, typically at the scale Λ .

³The normalization is computed at run-time based on the user’s choice of weight function and associated parameters. The user can also choose the range or interval of M_X over which the weight function has support, *i.e.* is non-zero, allowing for complex, “interrupted” weight distributions.

- Gaussian
- Bifurcated Gaussian
- Breit-Wigner resonance lineshape
- Flat or “box-like”
- “Dominating”—the weight function consumes as much of the inclusive rate as is required to meet the required partial width. The weight function thus follows the inclusive spectrum exactly over some interval $[M_0 - \delta M_1, M_0 + \delta M_2]$.⁴

Weight functions for different modes are allowed to overlap (except for the “dominating” weight function which allows no competition) to allow different exclusive channels to compete for rate from the same part of the inclusive spectrum.

To clarify the use of the weight functions, and their role in executing a particular $B \rightarrow X_u \ell \nu$ decay, we now describe the event generation sequence in detail.

A.3.1 Making the Decay Decision

At the abstract interface level, every inclusive “model” is expected to be able to provide a kinematic triple (E_ℓ, M_X, q^2) distributed according to the internal dictates of the model. This assumption is essential to the core InclGen decay engine. Note that it is a simple matter to accumulate a one-dimensional hadronic mass spectrum $d\Gamma/dM_X$ by simulating a large number of such triples.

Event generation begins with the request from QQ to decay a B meson with some specified kinematics. First, a kinematically feasible point (E_ℓ, M_X, q^2) in inclusive phase space is selected randomly, according to the distribution predicted by the inclusive model. Then, using only the hadronic mass information generated as part of the kinematic triple, a decision is made to either decay the hypothetical B into a nearby exclusive resonance, or to implement the decay non-resonantly.

The decision is made by comparing the inclusive rate $d\Gamma/dM_X$ at the generated value of M_X to the sum of the partial rates assigned to nearby exclusive resonances via the weight functions described earlier; recall that each of these essentially smears the true mass distribution for each resonance by an inclusive “smearing scale” Λ . The inclusive/exclusive decay decision is made in a random probabilistic fashion by Monte Carlo dice-throwing using a pseudorandom number generator.

⁴This weight function was introduced to deal with the π resonance, which lies at such low hadronic mass that there is essentially no inclusive rate available. The “greedy” nature of this weight function allows it to consume as much inclusive rate as possible at low mass until its partial width is satisfied.

The result of the dice toss is constrained to fall into the interval bounded by zero and the total inclusive rate available at the selected hadronic mass; the mode whose weight function is larger than but closest to the dice toss is selected as the “winner.” The initial kinematic triple is discarded and an exclusive decay handler is directed to generate a $B \rightarrow X_u \ell \nu$ decay for the selected mode X_u . If the dice toss exceeds the sum of all exclusive weight functions, the decay is instead passed to a non-resonant decay handler, and the initial kinematic point is preserved. Lepton flavor (e or μ) is also selected randomly, with equal probability.

A.3.2 Completing the Decay

Non-resonant Decays

If the decay is to be non-resonant, it is handled by standard Lund-like [84] CLEO software routines for hadronizing a $u\bar{q}$ system of mass M_X . The previously generated lepton energy E_ℓ and lepton-neutrino invariant mass q^2 are sufficient to determine the remaining kinematics of the system.

The `decqrk` and `phsp` routines in the Fortran `qqlib` library shoulder most of the work in carrying out the decay. Past CLEO measurements of hadronic multiplicities in $B\bar{B}$ events are used to generate the number of final-state particles, which are then grouped into mesons and baryons by the `decqrk` routine, consistent with the initial flavor of the parent B meson. Simple non-resonant phase space is used to determine the kinematics of the daughter particles thus selected.

Exclusive Decays

If the decay is to be along a particular exclusive channel selected randomly from those “near” the initial M_X value, the generated kinematic triple is forgotten and a new decay is carried out according to the ISGW2 model and the form factors appropriate for the selected resonance. The actual implementation is handled by the EvtGen package [86].

Finishing the Decay

The InclGen package internally interfaces to EvtGen for decay of the exclusive resonances, and requests that EvtGen attempt to decay the primary hadronic daughters as far as possible. Note that this is in contrast to the usual EvtGen behavior, where after a requested decay is executed, the decay products are immediately surrendered back to QQ for further decay. By allowing EvtGen to carry out later stages of the decay, we preserve angular correlations deeper into the decay

chain. (Recall that QQ does not track such information.) Similarly, any hadrons created in the non-resonant decay are also handed to EvtGen for further decay.

As configured for use with InclGen, EvtGen terminates its decay-handling when it arrives at a scalar particle; these decays can be handled with the simpler phase space-based routines employed by QQ with no loss of information since there are no angular correlations left to worry about.

After EvtGen has completed the decay, the system is boosted from the B rest frame back to the lab frame and all decay products are returned to QQ for final completion of the event generation cycle.

A.4 Sample Results

As should be clear, a chief advantage of the hybrid technique is that the non-resonant background shape respects the existence (and location) of known exclusive resonances, but it is also to some extent dependent on the shape predicted by theory in the inclusive limit of sufficiently broad averaging. Rather than being finely tuned for a few of the best-known resonances (*e.g.* π and ρ), this mixed approach creates a cocktail of many resonant and non-resonant channels that respects input from both experiment and theory.

Fig A.1 shows distributions of various kinematic quantities in a sample of 0.6 M B decays generated with InclGen. In each plot, the total generated spectrum is broken into non-resonant and resonant portions, and the sum is compared to the inclusive spectrum predicted by HQET.

A.5 Interface

The InclGen package offers users a high degree of control over its behavior, from the description of the masses, partial rates, and weight functions for each exclusive mode to the choice of inclusive model and its associated parameters. Most of these features are accessible either through FFREAD cards read in the QQ control file, or are read in at initialization from separate InclGen-specific definition files.

A.5.1 Parameters

Here we list the various cards that InclGen will recognize in the QQ control file. These are defined on the Fortran side in the usual file `qqinpt.F`, a special version of which is included in the InclGen package. All of these parameters have default values, as noted below, and many need not appear at all for the package

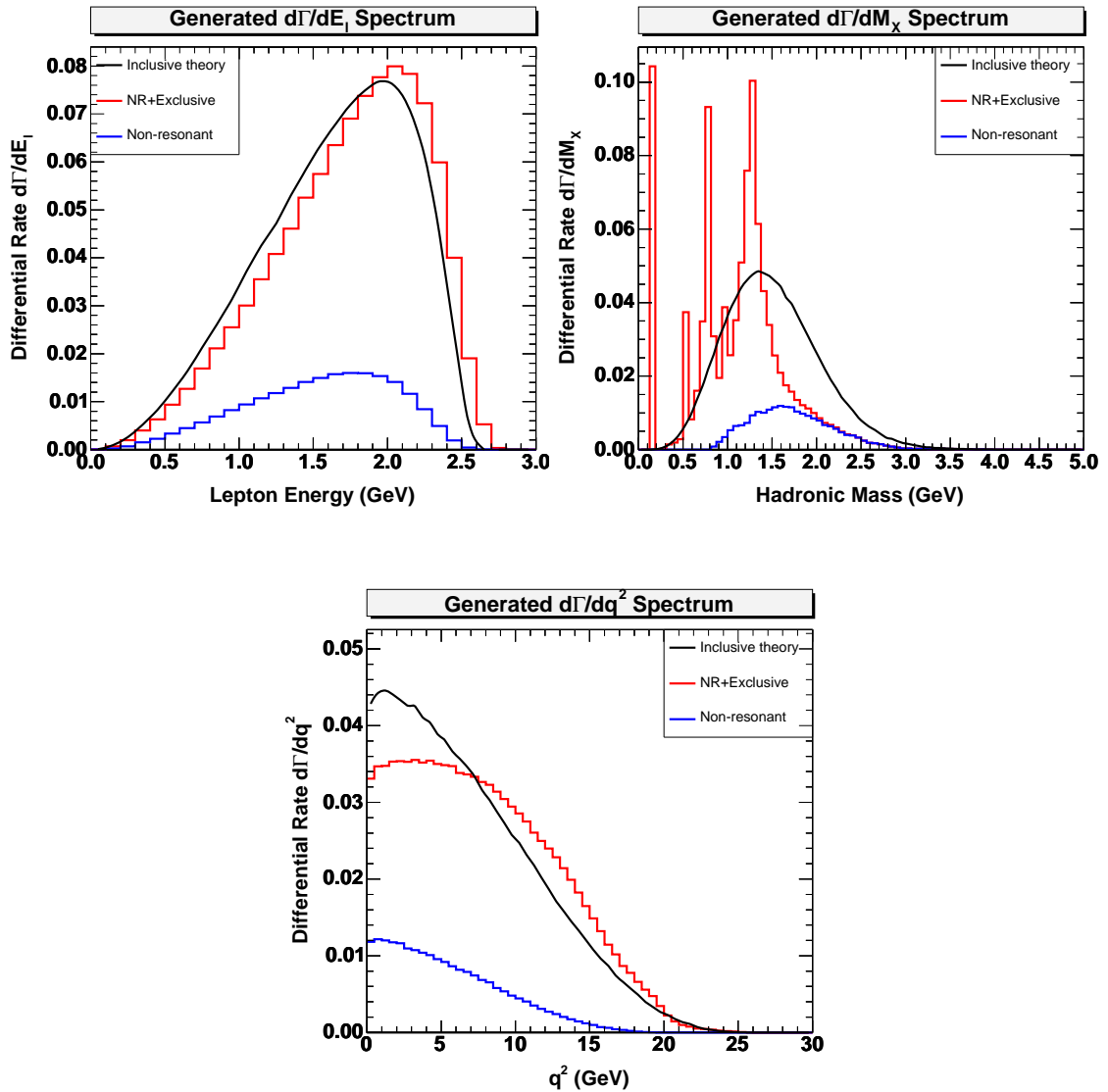


Figure A.1: One-dimensional spectra for events generated with the InclGen inclusive generator, for a sample of 666 $K B$ decays. The theoretical spectrum as predicted by HQET is indicated by the smoothed curve, and the non-resonant and ISGW2 resonant components as actually generated are represented by the two (stacked) histograms; the lower histogram is the non-resonant contribution. It is clear that the theoretical spectrum is ignorant of the resonant structure at low hadronic mass M_X , but the hybrid approach provides a systematic method for “borrowing” rate from the lower half of the inclusive spectrum that is then assigned to these resonances. The remaining inclusive rate is dedicated to a relatively smooth non-resonant contribution at higher hadronic mass.

to run. Parameters for unused models may be specified in the control file with no ill effect.

- **INCMDL**
A character string specifying the name of the inclusive model. Valid choices at present are “**ACCMM**” for the ACCMM model, “**HQET**” for the HQET-based model described above, and “**WeakAnn**” for the special model used for simulating weak annihilation in $b \rightarrow u \ell \nu$ decays. Default is “**ACCMM**”.
- **INCDEC**
Name of the file that describes the exclusive modes to be considered in the simulation. Path specifications can appear in the name. Default is “**inclgen.dec**”.
- **INCPATH**
Path for input and output files for InclGen, potentially useful when running the executable in a batch environment. Defaults to the current directory, “**./**”.
- Parameters for ACCMM model:
 - **PFERMI**
The value of the p_F parameter, which sets the scale of the energy carried by the light degrees of freedom in the B meson. Default is 0.300 GeV.
 - **MSP**
The mass of the spectator quark in the B meson. Default is 0.150 GeV.
 - **MLIGHT**
The mass of the light u quark produced in the weak $b \rightarrow u$ decay. Default is 0.150 GeV.
- Parameters for the HQET model
 - **LAMBAR**
The $\bar{\Lambda}^{\text{SF}}$ parameter for the shape function used to implement the non-perturbative smearing in the HQET-based model. Default is 0.480 GeV. Note this is *not* the central value from the CLEO $b \rightarrow s\gamma$ analysis!
 - **LAMONE**
The λ_1^{SF} parameter for the shape function. Default is -0.300 GeV². Note this is also *not* the central value from the CLEO $b \rightarrow s\gamma$ analysis!
- Parameters for the WeakAnn model
 - **QCDSCL**
The Λ parameter in the soft PDF that sets the slope of the exponential rolloff above the cutoff x_0 . Default is 0.300 GeV.

- CUTOFF

The x_0 parameter in the soft PDF; it sets the threshold below which the PDF is flat. Default is 0.500 GeV.

- WACLEP

A flag selecting the angular distribution to use in the simulation of the lepton-neutrino system in WA decays. The possible values are

0: flat

1: $\sin^2 \theta$

2: $(1 + \cos \theta)^2$

3: $(1 - \cos \theta)^2$

where θ is the angle $\theta_{W\ell}$ between the lepton flight direction in the W rest frame and the W direction in the B frame. Default is 0.

A.5.2 Running InclGen at CLEO

From the QQ user's point of view, InclGen is simply another "decay model" that handles a whole set of inclusive decay modes of the form $B \rightarrow X_u \ell \nu$, $\ell = e, \mu$. The library is linked into a (almost) standard qq executable that is driven by a standard control file, with the addition of a few control cards as described in the previous section. A qq-in-cleog executable can also be built quite easily to carry out physics and detector simulation at the same time.

To run the InclGen executable, several supporting files are required, as detailed below.

- **inclgen.ctr**

The control file that drives the event generator QQ. See the previous section for the control cards specific to InclGen that may be specified in this file.

- **decay.dec**

This file is the standard QQ file defining all possible particles and their available decay modes. The one packaged with InclGen includes the addition of the $a_0(1450)$ resonance discussed earlier, required if this particle appears in the `inclgen.dec` file listing the various exclusive modes. It is recommended that this version of `decay.dec` be used when running InclGen.

- **decay.evt**

The decay definition file for EvtGen. It contains information similar to QQ's `decay.dec` file, but also specifies the particular model to be used in carrying out decays along each channel. For instance, EvtGen handles all semileptonic B decays according to the ISGW2 prescription. Note that the branching fractions in this file are overruled by those in `decay.dec`. The version of this

file packaged with InclGen includes the addition of the new $a_0(1450)$ particle as well. It is recommended that this version of `decay.evt` be used when running InclGen.

Note that since EvtGen is given free reign to decay the X_u system with abandon according to its own decay tables, some new parts of the EvtGen-QQ boundary are explored. In particular, some residual differences in the decay channel specifications between QQ and EvtGen lead to slightly different decays than would appear if the X_u decay were handed immediately over to QQ.⁵

- `BTOU.dec`

This is the so-called “user decay file” for QQ that over-rides definitions in the standard decay file `decay.dec`. It is in this file that the channels “CHANNEL 92 0.00500 *UU* E- NUEB” etc. are actually added to the B decay block with a pseudo-branching fraction of 0.5% per lepton flavor.

NOTE: Without the use of this file or one similar to it, the InclGen algorithm will never be called and no inclusive $B \rightarrow X_u \ell \nu$ decays will be generated! Also note that the QQ model number for InclGen is **92**, one more than the 91 assigned to EvtGen.

- `inclgen.dec`

The exclusive mode definition file for InclGen. The actual name of this file is specified by the `INCDEC` card in the QQ control file. For a more detailed description of this file, see Sec A.6.2.

- `HQET_*.txt`

HQET model only: To avoid duplication of effort when simulating large amounts of data in parallel, InclGen typically expects to read in the four 1-D distributions $d\Gamma/dE_\ell$, $d\Gamma/dq^2$, $d\Gamma/dM_X$, and $d\Gamma/dM_X^2$ from external histogram files, rather than computing them at run-time using Monte Carlo integration.⁶ Since it is wise to avoid the independent use of `hbook` when running `qq` in `cleog`, the standard format for these histogram files is actually ASCII text, one line per bin. A cubic-spline interpolation is done at run-time to create the smooth 1-D distribution used during event simulation.

By running InclGen in a special initialization mode, it is possible to actually create these spectra at run-time and store them for use in subsequent running.

⁵These differences are not fatal, and amount to distinctions between decay channel specifications of $K_S K_L$ instead of $K^0 \overline{K}^0$ and other largely irrelevant details.

⁶This feature is really an artifact of interface design. Some inclusive models are best-suited for generating complete events, one at a time, and cannot deliver the 1-D marginal spectra directly. The easy way out is to generate a large sample of kinematic triples in a separate job and fill histograms to record the lower-dimensional distributions. These distributions can be then be used in subsequent event generation jobs. For instance, the $d\Gamma/dM_X$ distribution is required for making the inclusive/exclusive decay decision.

The naming convention for these files, when they are required, is based on the name of the inclusive model, *e.g.* `HQET_Mx.txt`.

NOTE: Because the $d\Gamma/dM_X$ distribution is actually used in the inclusive/exclusive decision algorithm, it is important to keep these 1-D spectra files up-to-date relative to the model parameters specified in `inclgen.ctr`. That is, if either of the parameters `LAMONE` or `LAMBAR` are modified, then new `HQET*.txt` files need to be made before new events can be properly generated.

The final `InclGen/qq` executable is run just like the default `qq` executable: the name of the control file is supplied at run-time, and event generation proceeds according to the control cards provided therein. The output is typically a `*.rp` (roar) file, or a `*.fzx` file if `cleog` is being run in tandem.

`InclGen` decays can be distinguished at run-time by the unique model number 92 assigned to `InclGen`. In the output file, the $b \rightarrow u \ell \nu$ decays can be similarly identified, or by checking the decay channel number (which can be reconstructed from the particular pair of standard and user decay files used to generate the Monte Carlo sample), or by simply iterating over the B daughters to determine empirically whether the decay is semileptonic and charmless. This last method is considered more general and more foolproof, since it is robust against other changes in the input decay definition files that may change the channel number assigned to the `InclGen` decay mode(s).

A.6 Implementation Notes

We focus here on a few of the relevant decisions and design details of the `InclGen` package that may be of use to those working with the code in the future.

A.6.1 Link to QQ

The link between `QQ` and `InclGen` is modeled on the similar calling mechanism designed to link `EvtGen` into `QQ`. That is, the driving routine `qqdeca.F` has been modified to call the `inclgenlink.F` routine when the matrix element or model number 92 is found for a (B) decay mode. This custom code is simply a Fortran wrapper that calls the the C++ routines in the `InclGen::` namespace necessary to generate a single $B \rightarrow X_u \ell \nu$ decay.

The `InclGen` package initializes itself when the first inclusive $b \rightarrow u \ell \nu$ decay is requested.

Communication with the internal particle lists of `QQ` involves common block manipulations that are best left undescribed. That is, rather than making likely

false claims here about how the communication is actually implemented, it is best for interested parties to survey the code themselves and hazard their own guesses about how and/or why the code works. At all.

A.6.2 Definition of Exclusive Weight Functions

The exclusive modes available to InclGen are defined in the `inclgen.dec` file described above in Sec A.5.2. This file defines a block of modes for each of the four types of B mesons, listing the properties of the hadronic resonances available to the particular parent B , as well as the associated weight function for each mode. The parser for this file ignores extra whitespace, blank lines, and any lines that begins with the “#” character.

To understand the structure of this file in more detail, we can examine the first few lines, as shown in Fig A.2.

```
# Each channel is described by:
# Width Had    Lep  Neu  Mhad  Wid    WghtFn  FnWidth OtherParams

Decay B0
0.960 PI-      E+   NUE  0.1396 0.0001 DomAuto  1.250
1.420 RHO-     E+   NUE  0.770  0.151  Gauss   1.750  0.00 4.00
0.050 A1450-   E+   NUE  1.474  0.250  Gauss   1.250  0.00 100.00
0.870 A1-      E+   NUE  1.230  0.400  Gauss   1.250  0.00 100.00
0.330 A2-      E+   NUE  1.318  0.104  Gauss   1.250  0.00 100.00
1.090 B1-      E+   NUE  1.230  0.142  Gauss   1.250  0.00 100.00
0.170 PI2S-   E+   NUE  1.300  0.300  Gauss   1.250  0.00 100.00
0.410 RHO2S-  E+   NUE  1.465  0.310  Gauss   1.250  0.00 100.00

0.960 PI-      MU+  NUM  0.1396 0.0001 DomAuto  1.250
1.420 RHO-     MU+  NUM  0.770  0.151  Gauss   1.750  0.00 4.00
0.050 A1450-   MU+  NUM  1.474  0.250  Gauss   1.250  0.00 100.00
0.870 A1-      MU+  NUM  1.230  0.400  Gauss   1.250  0.00 100.00
0.330 A2-      MU+  NUM  1.318  0.104  Gauss   1.250  0.00 100.00
1.090 B1-      MU+  NUM  1.230  0.142  Gauss   1.250  0.00 100.00
0.170 PI2S-   MU+  NUM  1.300  0.300  Gauss   1.250  0.00 100.00
0.410 RHO2S-  MU+  NUM  1.465  0.310  Gauss   1.250  0.00 100.00

Enddecay
```

Figure A.2: Sample block from the `inclgen.dec` file, defining the exclusive modes available for InclGen for semileptonic charmless decay of the B^0 . Mass, width, and weight function parameters are all in units of GeV. The first number for each mode defines the partial width for the channel, in units of 10^{-13} s^{-1} , modulo the unknown factor of $|V_{ub}|^2$.

This decay block defines the eight different X_u resonances available for $B^0 \rightarrow X_u e \nu_e$ decay, declaring the name and mass of each hadronic resonance and then describing its exclusive weight function as a simple Gaussian with some nominal width $\Lambda \sim 1.25$ GeV centered on the nominal mass for the resonance. The additional parameters list the range in M_X over which the weight function is defined for normalization purposes; the interval $[0, 100]$ is essentially unbounded on the high-side, but includes a non-negative low-side cutoff. The π modes employ the special `DomAuto` mode that computes, at run-time, the limits of the exclusive weight function from the inclusive spectrum such that all of the B^0 decay rate is split between the $\pi e \nu_e$ and $\pi \mu \nu_\mu$ modes equally at low hadronic mass, with no competition from the other modes (despite the fact that their support intervals are initially declared to be $[0, 100]$).

Modifications to this file are easy to make, but must be consistent with the requirement that at all positive values of the hadronic mass M_X , the requirement of “unitarity”

$$\sum_i \left(\frac{d\Gamma}{dM_X} \right)_{i \text{ excl}} < \left(\frac{d\Gamma}{dM_X} \right)_{\text{incl}} \quad (\text{A.6})$$

must be satisfied; that is, the sum of the exclusive weight functions cannot exceed the total inclusive rate available.

Fig A.3 shows an (old) example of how these weight functions are used to carve up the inclusive rate for the purposes of making the inclusive/exclusive decay decision.

A.6.3 Code

Except for the interface to QQ, and some low-level routines borrowed from the `qqlib` and CERN code libraries, the entire `InclGen` package is written in C++. A quick survey of the inheritance hierarchy should help newcomers to understand the (minimal) organization behind the various classes and types.

- **InclGen**
This is a static class that serves basically as a namespace for the external hooks to start and run the various parts of the `InclGen` package, chiefly: event generation.
- **InclusiveGenerator**
This singleton class coordinates model initialization, parameter selection, and event simulation, including the inclusive/exclusive decay decision.
- **InclModel**
The base class for the different inclusive “models”; it defines the interface for generation of an event consistent with the internal constraints of a generic

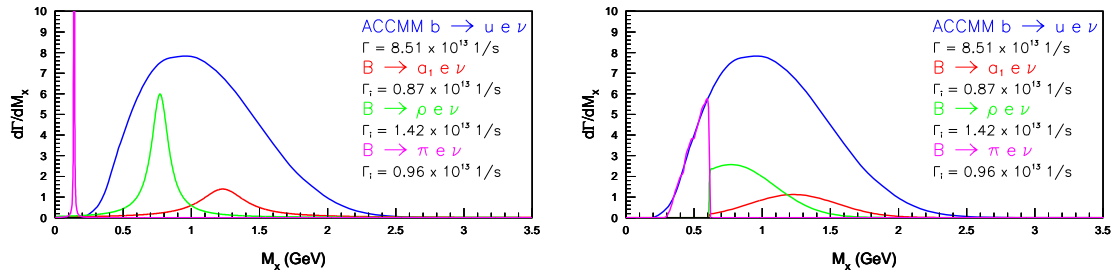


Figure A.3: Sample “weight” functions assigned to a few exclusive modes, for a nominal version of the ACCMM inclusive model. On the left is the true mass spectrum for the π , ρ , and a_1 resonances, drawn as Breit-Wigners with approximately the natural width for each. (The π width has been exaggerated slightly for clarity.) On the right is shown a sample division of the inclusive spectrum into pieces assigned to each exclusive mode, essentially “smearing” out the exclusive rate on a scale $\Lambda \sim 500$ MeV. The amount of rate dedicated to each is determined by the ISGW2 partial rate, specified in the `inclgen.dec` file. The weight functions are tuned slightly by hand to ensure that their sum never exceeds the inclusive envelope. We emphasize that the weight functions simply indicate where the rate assigned to each exclusive resonance is drawn from; they do not indicate the actual mass distribution for each channel. For comparison, the ACCMM inclusive spectrum is drawn in blue in both plots.

inclusive model, incorporates various sanity checks on the kinematics, allows for parameter retrieval, etc.

- **DecayHandler**
Base class for carrying out any kind of semileptonic B decay.
- **InclDecayHandler**
This class handles the hadronization and kinematics for non-resonant decays
- **ExclDecayHandler**
This class handles decay along a selected exclusive channel.
- **ExclDecayChannel**
Base class for representing a single exclusive decay channel for a particular B and lepton flavor. Includes definition of general functionality for support of the notion of a “weight function” defining the range of M_X over which this mode competes for a portion of the available inclusive rate.
- **ExclDecayList**
Container class for all of the exclusive decay channels available to a particular flavor of parent B ; used by `InclusiveGenerator` in making the inclusive/exclusive decay decision.

Various other classes (`IGRandom`, `IGDefinitions`, `ParticleNames`, etc.) help abstract the design and organize ancillary parts of the package.

A.6.4 Compiler Flags

A few compiler flags are managed by the general `IGOptions` class. The status of these flags is reported at run-time when `InclGen` initializes. The various options are:

- **INCLGEN_USE_HBOOK**
This flag determines whether any histogram manipulation or file retrieval uses CERN's `hbook` package or the limited functionality of the home-grown `BTHistogram` class, part of the `BagOfTricks` library; see Sec A.6.5.
- **INCLGEN_DEBUG_HIST**
When this flag is set, `InclGen` stores an ntuple with the complete kinematics of the first n events, where n is set by the related `INCLGEN_NTUPLE_SIZE` flag. The ntuple is only saved if the `INCLGEN_USE_HBOOK` flag is set. The various exclusive weight functions are also stored in the output histogram file for later reference.
- **INCLGEN_RECALCULATE_MAX**
When this flag is set, the code steps systematically through the three-dimensional $b \rightarrow u\ell\nu$ phase space, searching for the maximum of the triple differential rate. The maximum is used when drawing kinematic triplets from the distribution. If the flag is not set, a conservative hard-wired maximum is used instead.
- **INCLGEN_RECALCULATE_RATE**
When this flag is set, the total inclusive rate is re-computed by using Monte Carlo integration. This step also creates the various 1-D spectra needed later in full event generation, namely, $d\Gamma/dM_X$. If the flag is not set, the spectra are read in from histogram files (text or `hbook`, depending on the other flags).

Several of these options can be specified at compile-time by appending them to the `InclGen.defs` file in the base directory of the package. These entries in this file are concatenated to the compiler commands issued by the `make` facility.

A.6.5 Other Libraries

The `InclGen` executable relies on a few supporting libraries for successful operation. It makes use of the following:

- **BToUTripleDifferential**
This library contains the actual implementation for the calculations of the triply differential rate in the HQET-based model. It is part of the CLEO III software repository.
- **EvtGen**
Exclusive decays are handled by EvtGen, as are all secondary decays of the primary hadronic daughters of resonant or non-resonant decays. This library has been slightly modified to cooperate with InclGen in a friendly and more uniform fashion.
- **qqlib**
The interface to qq as well as various parts of EvtGen and the non-resonant decay routines rely on code stored here. This library has been slightly modified from the standard CLEO II version to repair a few minor bugs which frustrated smooth operation.
- **QQLib**
This small library defines a uniform C++-wrapped interface for many of the lower-level qqlib routines actually used in InclGen. This library is available where the InclGen package is stored.
- **QQCommons**
Provides C++ access to various Fortran common block used by QQ. This library was created specifically to support InclGen and is available where that package is stored.
- **BagOfTricks**
This library contains various helper classes to ease operation of InclGen, including input/output classes, a function minimization interface, a toy histogram class, a cubic spline class, and others. This library was created specifically to support InclGen and is available where that package is stored.
- Various CLEO III software libraries, such as the histogram interface, Fortran interface, ToolBox, Utility, lunmgr, CLHEP, and the CERN software libraries packlib and mathlib.
- The bulk of the usual CLEO II libraries, as listed in the package's Makefile.

A.7 Future Improvements

Although extremely functional, the InclGen algorithm can still benefit from additional improvements. More important than a simple design or interface change is the incorporation of improved physics modeling. One element so far neglected is the use of q^2 information in making the inclusive/exclusive decay decision. It

should be possible to use the doubly-differential distribution $d^2\Gamma/dq^2 dM_X$, compared to a similar and suitably smeared-out quantity for the exclusive modes, to determine how to decay the parent B .

No other improvements are on the foreseeable horizon, although many are certainly possible.

Appendix B

Identification of Hadronic Splitoffs

This appendix is a manual HTML \rightarrow L^AT_EX translation of the extant documentation for the `SplitoffProd` package referred to in the main text (Sec 5.3). The original document can be found in the `Doc/` subdirectory of the code module in the CLEO3 software system. It is also accessible online through the web interface to the CVS repository.

Acknowledgments

The development (design, coding, and testing) of this package was carried out by T. Meyer and N. Adam, under the guidance of L. Gibbons. The first version was released in late 2001; there have been only a few small updates since then. The simple maintenance and documentation responsibilities are currently still managed by Meyer.

B.1 Introduction

This documentation is intended only to be an informal user's guide to the new `SplitoffProd` package available in the CLEO3 software library. Some background material on the nature of splitoff showers and the concepts of neutrino reconstruction are provided, but the focus is not on the details of how the code works, but rather how to use it, what the user-level "knobs" are, and what confidence can be placed in the package's performance. An overview of the algorithm is included just to provide the user with a sense of what ingredients are used in the splitoff decision. The implementation notes are more of a crib sheet to understand how this package compares to the old one, but no attempt is made to fully explain the complete splitoff decision algorithm. See the references listed at the end for a more thorough discussion of neural net binning, the precise use of π^0 's, and so on.

B.1.1 Splitoff Showers

Showers in the calorimeter arise from both charged and neutral particles. Ordinary track-shower matching does a good job associating showers with the tracks that produced them by projecting the track out to the calorimeter face and looking for nearby showers. But hadronic interactions in the initial shower can often lead to secondary particles that travel some distance away from the primary shower

before depositing the bulk of their energy as a shower that is displaced or “split off” from the main one. These showers may not be near the initial track projection but should properly be associated with that particle and *not* considered as additional “neutral energy” in the calorimeter. Avoiding this kind of double-counting is critical to the success of neutrino reconstruction.

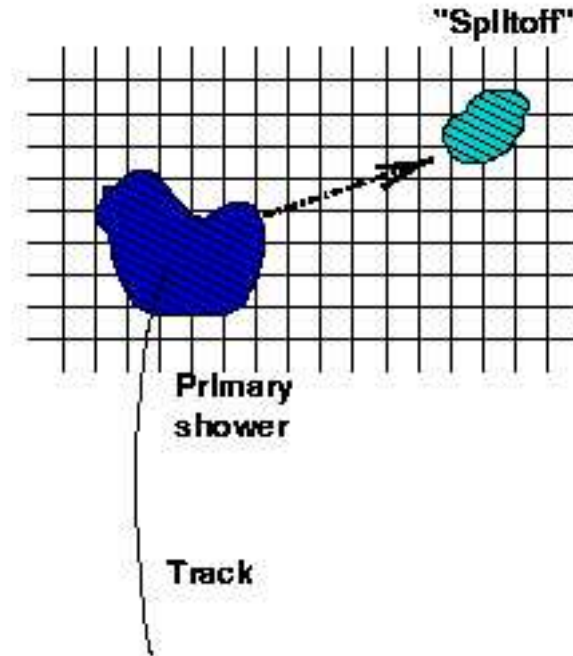


Figure B.1: Author’s rendition of the production of a splitoff shower in the calorimeter. An incident charged particle creates a primary shower on the left, and secondary particles travel along the direction indicated by the arrow before finally depositing their energy in a secondary “splitoff” shower, displaced from the primary shower. The energy the incident track deposited in the calorimeter is now divided between two separate showers. For successful neutrino reconstruction, one must be able to recognize that the second shower is actually related to the first, and so to the track as well.

Fig B.1 tries to illustrate the situation in cartoon-like fashion: an incident track creates a primary shower at one location in the calorimeter. Secondary particles created in that shower travel several crystals away before depositing the rest of their energy. This second shower is not contiguous with the first, but there are clues in its shape (orientation, distance, and pattern of energy deposition) that can be used to “trace back” the daughter shower to its parent.

B.1.2 Neutrino Reconstruction

Correctly identifying showers as splitoffs is crucial to the method of neutrino reconstruction. In this technique, one takes advantage of the hermeticity of the CLEO detector and identifies any deficit between the total observed energy and the initial beam energy with the kinematics of an undetected neutrino. The assumption is that all of the other particles in the event, charged and neutral, were observed and their (\vec{p}, E) properly reconstructed. Typically, to obtain the best resolution on the neutrino four-vector, one ignores the showers that are matched to tracks and instead takes the energy from the track's momentum and the mass hypothesis supplied by some particle identification algorithm. With this recipe, counting the energy of showers that are matched to tracks is redundant, since the particle energy is included in the loop over tracks. Clearly, accurate neutrino reconstruction relies on correctly accounting for each particle's energy *exactly* once. If the energy of a splitoff shower is counted in addition to the energy of the primary track that caused it, the double-counting will bias the neutrino energy estimate. In order to be successful, then, neutrino reconstruction analyses must identify splitoff showers so they can be properly discarded. But for similarly obvious reasons, the procedure must not discard real photons too often, either!

Splitoff identification can also be useful as a general indicator of shower quality, or for screening showers for use as photon candidates, etc. The general user is cautioned, however, to consider carefully whether the criteria under which this package was developed (neutrino reconstruction) really apply to their own particular analysis.

B.1.3 Historical Interlude: Splitoff Identification at CLEO

Lawrence Gibbons created the original `splitf` package in 1994 to identify and reject splitoff showers in CLEO II data as part of an effort to cut down on the amount of misidentified neutral energy in the calorimeter. This code made its big debut in the 1996 observation of exclusive $b \rightarrow u \ell \nu$ in CLEO II data with the then-new technique of neutrino reconstruction. Having passed intense scrutiny at that time, his package has now become a de facto standard in subsequent neutrino reconstruction analyses.

The `splitf` package has since been used, for instance, in the recent analysis of the moments of the hadronic mass spectrum (in semileptonic B decay), the new II+II.V exclusive $b \rightarrow u$ analysis, and in parts of the recent $b \rightarrow s \gamma$ analysis. As a powerful analysis tool, the splitoff rejection package will be of equal or even greater value in future rounds of CLEO III and CLEO-c analyses.

However, `splitf` was written in Fortran to run within the CLEO II analysis

framework called `Driver`. It made use of `zfiles`,¹ shower cell lists, and other outdated quantities no longer appropriate or even available for an analysis based in the newer, more flexible environment of `suez`. Hence, to carry neutrino reconstruction forward into the CLEO III era and beyond, we’ve developed the `SplitoffProd` Producer, which re-implements the old algorithms in a generic way that can be applied to analysis of CLEO II, CLEO III, and CLEO-c data. The new package exhibits the same performance on CLEO II data as the old, but can process CLEO III data as well. Hence it serves as both a replacement and extension of the original package.

B.2 Using `SplitoffProd`

This section is intended to describe the *Splitoff* interface in just enough detail for casual users to get splitoff identification working and incorporated into their own analysis. Subsequent sections provide more details on the design and operation of the code, including how to influence its default behavior and how to assess systematics.

B.2.1 What it Feels Like

Before turning to code examples and syntax directives, let’s first examine the most common way in which users are expected to interact with the results of splitoff identification.

The `SplitoffProd` Producer analyzes each `NavShower` in an event and assesses whether it is a likely hadronic splitoff or otherwise unsuitable for use in reconstructing the neutral energy in an event. (For instance, showers that are rather isolated are still tested to see if they seem to really “point” back to the origin. Minimum energy cuts and track-matching requirements are also imposed.)

It provides a list of `NavShowers` that are “approved” for such use, as shown in Fig B.2. One needs simply to loop over the list of approved showers on the right (in green), instead of looking up the splitoff decision for each shower in the event to see if the shower was “approved.” With the awesome power of the “usage tag” in the `suez` extract call, you can simply ask for the list on the right in one line and get it with no additional work on your part.

¹`zfiles` was an arcane database developed for storing CLEO II constants and geometry information. Experts on its design and use were few and short-lived, making compatibility in the CLEO III era a hopeless goal. Rumor has it that part of the reason for deciding to write new software for the CLEO III detector upgrade was to get away from dependence on this creaking database system. . . .

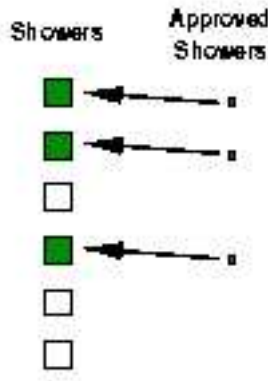


Figure B.2: Another cartoon, this time of the relationship between the list of all reconstructed showers in the event, and the subset of showers that have been “approved” for use in neutrino reconstruction. On the left is the “table” or list of all showers; each box represents a shower object. The green ones are those that have been approved for use in neutral energy calculations. On the right is simply a list of pointers to these same shower objects, with entries only for the approved showers.

Briefly, to use the Producer, you need to have `NavShowers` available in the Frame. When running on Pass2’ed data, this simply requires using the script `runOnPass2.tc1`, as shown in Sec B.2.4. (See the later Sec B.5 to see how this is done when running on CLEO II data.) If for some reason you’re running on raw data, you will have to run shower reconstruction on your own. (Details for that case are *not* described here.)

You will also need to consider choices for particle ID and whether you wish to limit what tracks are used in track-shower matching.

B.2.2 Particle ID

Splitoff obviously uses track-shower matching information to identify which showers are *un*-matched, and therefore are splitoff candidates. But it also uses the *species* (e , μ , h) of each matched track to assess how likely it is that the given track actually produced a splitoff shower near the matched shower. For instance, muons produce essentially no splitoffs, high-energy electrons produce very few, while hadrons are reasonably likely to do so across a wide range of energies. So splitoff likelihood depends in part on the identity of the track matched to the parent shower. Hence ***Splitoff* requires particle identification information**. An early version of the *Splitoff* package had simple particle ID built right into it, but this quickly proved too inflexible. Neutrino reconstruction analyses in particular

spend a lot of effort on getting particle ID right; it only makes sense that *Splitoff* shouldn't use its own (and possibly different) guesses about the identity of each relevant track. The interface now allows the user to employ their own PID, and simply deliver lists of electrons and muons to the Frame for *Splitoff* to use. By extracting these lists, *Splitoff* can easily determine the species of a track as either muon, electron, or neither, *i.e.* some sort of hadron.

The details of this mechanism are simple: *Splitoff* extracts a list of `NavTracks` with usage tag "Muons" and another with usage tag "Electrons". For each track that is matched to a shower, *Splitoff* essentially looks up the particle species by finding in which list that track appears. If it's not in either list, the track is internally labeled a generic hadron for the purposes of splitoff identification.

The use of the *Splitoff* package thus *requires* particle identification information for every track in the event, or at least for all of those that are matched to showers. To supply this information, users can select the existing `SimpleMuonIdProd` and `SimpleElectronIdProd` Producers, which fill these lists using Brian Heltsley's "simple" lepton identification algorithms. (Read the header files `NavMuonId.h` and `NavElecId.h` to see what the lepton identification requirements are.) Alternatively, users can copy or modify these Producers to incorporate their own electron and muon ID. The use of "selection" Proxies makes this an easy and straightforward task. Look at the code in the these Producers for a simple, working example.

Users aren't restricted to delivering these lists with the fixed usage tags "Electrons" and "Muons", either. If desired, one can specify alternative usage tags for the lists of tracks that *Splitoff* will treat as electrons and muons with the `suez` Parameters `usageTagForMuons` and `usageTagForElectrons`.

Note that it is important that these two particle ID lists be kept disjoint and non-overlapping. The code by default searches the muon list first, but it is not wise to rely on this behavior. For this reason, the `SimpleElectronIdProd` Producer actually extracts the list of muons (the usage tag here can also be altered from its default "Muons" with a Parameter change) and only makes an attempt to identify as electrons those tracks that are not already identified as muons. This precedence of muon ID over electron ID makes sense given that electrons really don't ever fake muons, while the opposite is somewhat more feasible.

B.2.3 Tracks for Track-Shower Matching

Sometimes tracks are identified in an event and then extrapolated to the calorimeter face for matching to showers when it's less than certain that a real particle actually flew along that path. For instance, a hard scatter in the inner part of the detector might lead to two distinct tracks, one fitting the hits from the beamspot to the point of the scatter, and one following the particle after that point. The latter track is suitable for use in track-shower matching, but the inner track doesn't

describe a real particle at the point where it's projection finally reaches the crystals.² Any shower this projection intersects should not really be considered as "matched." Since an important part of the *Splitoff* algorithm is in identifying the showers that might have splitoffs nearby, critically reviewing the track-shower matches found by the default matching routine is important.

In the days of CLEO II, the Trkman package took care of identifying curlers, ghosts, and other tracking pathologies. It also provided a list of tracks that were suitable for use in track-shower matching. The new *Splitoff* package is designed to utilize a similar list, and retrieves it by extracting a list of `NavTracks` to be considered when reviewing matches. A shower will be considered "matched" only if the matching track appears in this list. The user can specify the name of this list (*i.e.* its usage tag) via the `suez` Parameter `usageTagForMatchTracks`. The default value is the empty string "" meaning that *all* tracks are considered usable for matching.

Note that *Splitoff* does not *create* new track-shower matches. Rather, it reviews all of the default matches (made from all tracks and all showers with relatively loose criteria) and drops those that don't meet its matching criteria. Specifically, any match to a track that's absent from the user-specified list will be dropped.

B.2.4 What To Do

Eager users will be anxious to get to the bottom line. The boxes below show the relevant lines to be added to your Tcl script to add the *Splitoff* package to your analysis.

In your tcl script

```
# Load Producers for running on Pass2'd data
run_file $env(C3_SCRIPTS)/runOnPass2.tcl

# Use "simple" electron and muon ID
prod sel SimpleMuonIdProd
prod sel SimpleElectronIdProd

# Identify splitoff showers
prod sel SplitoffProd
param SplitoffProd splitoffFilePath $env(C3_DATA)

## Throw out matches to tracks that are not Trkman-approved for
```

²Note also that the second track, on the other hand, would not be the best choice for use in neutrino reconstruction, where it's important to get a handle on what happened in the initial decay reaction, before any of the particles collided with detector material.

```
## use in trk-shwr matching
#prod sel TrkmanSelectionProd
#param SplitoffProd usageTagForMatchTracks "TrkmanTkShMatch"
```

Note that the `splitoffFilesPath` parameter *must be specified*: it gives the location of the lookup tables used as part of the splitoff rejection algorithm, including configuration and training data for the internal neural nets, carefully optimized cut values, and bad/good ratio lookup lists. The standard set of these files is located in the `Tables/` directory of the `SplitoffProd` module. These are now automatically installed into the `$C3_DATA` area of the software release when the package is built, but the Parameter must still be set to point there. The default is "", *i.e.* the current working directory.

Note the Producer will issue an error message and will `exit(1)` if you do not specify this parameter!

The minimum required Producers include `NavigationProd` to provide tracks and showers, `SimpleMuonIdProd` and `SimpleElectronIdProd` to provide the lists of "Muons" and "Electrons" tracks, and `PhotonDecaysProd` to find π^0 's (used in "protecting" showers that form good π^0 's from being identified as splitoffs too aggressively).

The *Splitoff* Producer (well, the Proxy it holds) is triggered by a request to extract `NavShowers` with the "SplitoffApproved" usage tag, as shown in the trivial example below. This code fragment is taken from the `event()` method of a "real-life" analysis Processor.

In your analysis code

```
// Extract list of showers to use for adding up
// true neutral energy
typedef FTable< NavShower > NavShowerTable;
NavShowerTable approvedShowerList;

extract(iFrame.record(Stream::kEvent), approvedShowerList,
       "SplitoffApproved");
```

You can then iterate over this list of `NavShowers` as usual, content that it contains just the "good" showers—those likely to be from real photons.

B.3 SplitoffProd Parameters

The package has several important Parameters, all documented briefly here. Remember that very terse descriptions are also available on the `suez` command

line with the usual command-line syntax:

```
param SplitoffProd <paramName> help
```

- **splitoffFilePath**

Described also above in Sec B.2.4, this Parameter specifies where the various cut and neural net initialization tables are stored. The Parameter is initialized to the empty string "" by default, and must be set to something sensible for the code to run. Using `$env(C3_DATA)` is the recommended choice.

- **usageTagForElectrons** and **usageTagForMuons**

Described above in Sec B.2.2 in some detail, these two strings specify the usage tags to use when extracting the lists (`FATable< NavTrack >`) of electrons and muons from the Frame. Recall these PID assignments are used when choosing the cut to apply to the neural net output. Since muons and electrons are less likely to leave splitoff showers in the calorimeter, the cut is modified for showers that are near showers matched to leptons.

Note that for your convenience (but not necessarily for your unconsidered use) there are two publicly available Producers, `SimpleElectronIdProd` and `SimpleMuonIdProd`, that produce such lists by using the simple electron and muon identification algorithms accessible directly through the `NavTrack` interface.

The Parameters defaults to "Electrons" and "Muons", respectively. Also, see note below in Sec B.9.1 that these lists are expected to be mutually exclusive. (The `Simple...IdProd` Producers mentioned above conform to this requirement.)

- **usageTagForMatchTracks**

Described above in Sec B.2.3 in more detail, this Parameter specifies the usage tag of a restricted list of tracks to use when identifying matched showers. Essentially, a track must be in this list in order for a match between it and a shower to be considered valid.

- **minimumShowerEnergy**

A new Parameter that allows user control over the minimum energy a shower must have in order to be “approved” as a real photon shower. The default is 25 MeV; below this energy, there’s a mixture of noise, unidentified splitoffs, and other unmatched showers from charged particles. Unfortunately, at such low energies, there isn’t enough information available to discriminate between these sources; it simply turns out best (for neutrino reconstruction) to drop them all. If you want to try lowering this cut, be aware that the nets and cuts were all developed with the current cut; the neural nets in particular have never seen a shower with an energy lower than 25 MeV.

The units are in GeV.

- **neuralNetBias**

Another new feature intended to provide users with a natural way to gauge just how sensitive their analysis is to the details of splitoff identification. See detailed description in Sec B.4 on systematics below. By default, this Parameter is 0, meaning no smearing is applied. Setting it to some small positive number turns on a random smearing of the neural net output that will change the splitoff identification results. One can then use this “detuned” version of `SplitoffProd` to determine how much one’s analysis depends on the inner workings of this algorithm.

Recommended value for systematics evaluation: 0.20.

B.4 Assessing Systematics for your Analysis

A good chunk of the success of neutrino reconstruction at CLEO has been due to the huge investments made in creating the original splitoff identification package. As such a critical element of those analyses, it is important to understand just how sensitive physics results are to the details of splitoff shower modeling in the Monte Carlo.

Even if the package is not very efficient or has a high rate for identifying fake splitoffs, as long as it performs identically on data and Monte Carlo, any sub-optimal performance does not act as a source of systematic error to an analysis. But if splitoff showers are not realistically-modeled in the Monte Carlo, blindly applying *Splitoff* can introduce unsuspected distortions or biases when the simulated data is used as a guide to interpret the real data.

For this reason, a new functionality was introduced into `SplitoffProd` in Sep 2004. There is now an option for the user to turn on a random “smearing” of the neural net output, which is applied before any cuts are made—before the actual splitoff “decision.” This degraded version of the neural net serves as a way of analyzing the systematic error inherent in using `SplitoffProd`. The effect that worsening the net’s discriminating power has on an analysis can be considered a measure of how much an analysis depends on accurate modeling of splitoffs in the Monte Carlo. As such: *This option is intended for use on Monte Carlo only*, since it serves as a way of stressing data-Monte Carlo agreement.

The smearing of the neural net result NN is implemented according to the following equation:

$$NN \rightarrow NN' = NN \pm s |G| |NN \pm 1|$$

Here, $s \geq 0$ sets the “smearing scale”, G is a Gaussian-distributed random number with mean 0 and variance 1, and the (+) sign applies to showers that

are tagged to photons, the $(-)$ sign for all others. The intent here is to degrade the performance of the net by smearing showers caused by real photons toward the splitoff end of the neural net range $(+1)$, and to smear all other showers (which are not, then, from real photons) in the photon-like direction (-1) . Note the smearing is random (G), scaled by s , and also proportional to how far the original net result is from the photon or splitoff end of the range as appropriate ($|NN \pm 1|$).

Fig B.3 is a comparison of the neural net output value before and after applying smearing to a generic $b \rightarrow u \ell \nu$ sample. This is the neural net “spectrum” for unmatched, “nonisolated” showers that are not tagged to photons; the smearing pushes the net values to the left, toward the photon end of the spectrum, as promised.

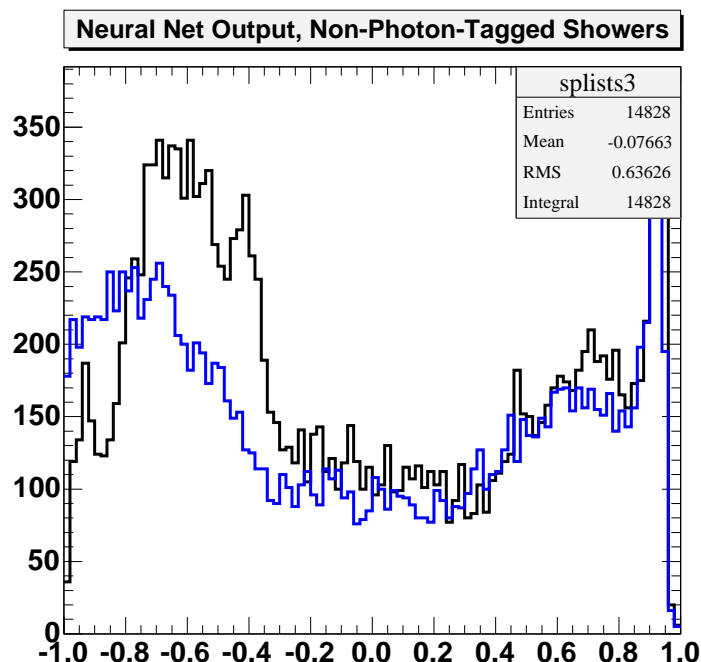


Figure B.3: The black histogram is the neural net distribution for unmatched, “nonisolated” showers for which neural net evaluation succeeded. The blue histogram shows the result after smearing the neural net output with $s = 0.2$.

In order to turn on this smearing, set the `neuralNetBias` Parameter to a non-zero value. (The recommended value is 0.20.) You will also need to select `MCTagHolderProd` to allow `SplitoffProd` to access tagging information through the `NavShower` interface. The Tcl snippet below shows this.

In your tcl script

```
# . . . Usual Producers, like above . . .
```

```

# Identify splitoff showers
prod sel SplitoffProd
param SplitoffProd splitoffFilePath $env(C3_DATA)
param SplitoffProd neuralNetBias 0.20

# Provide Nav access to tagging information
prod sel MCTagHolderProd

```

Note this discussion does not account for other lingering systematic errors inherent in the use of this package, such as discrepancies between the *number* of splitoff showers in data and Monte Carlo.

B.5 Usage for CLEO II Data

The `suez` interface includes modules for reading old-style CLEO II roar files, and Proxies have been written to fill the usual CLEO III analysis objects from the CLEO II common blocks. (Remember that in the Fortran world of Driver, these common blocks are filled from the roar fields of the data file by magical `GTXYZW` calls. These days are now over, at least as far as the `suez` user is concerned.) The package that wraps up all of this CLEO II access code is called `DriverDelivery`, since it delivers to the user all of the information once accessible in the Driver common blocks. The shared module produced from this package is `DriverSourceFormat`, which also links in the necessary CLEO II libraries to carry out its function. As shown below, selecting this source format is sufficient to “activate” all of the Proxies that will then create and properly stuff CLEO III objects when an analysis calls for them.

The *Splitoff* algorithm uses relatively high-level shower information, such as shape, energy, and position information; these are all properties that are contained in the `CcShowerAttributes` object, which can be filled from CLEO II or III data. Thus the *Splitoff* code doesn’t actually know where the showers it’s analyzing came from: it simply uses the track and shower objects it extracts from the Frame. The algorithm is abstract enough to be independent of whether shower energies originally came from a common block or a previously-stored `CcBasicShowerAttsArg` object. Hence the only change for running on CLEO II data is the use of `DriverSourceFormat` to provide the appropriately-created track and shower objects.

To use `SplitoffProd` on CLEO II data, simply include the following in your Tcl script instead of the above snippet.

In your tcl script

```

# We're running on CLEO II roar data
source_format sel DriverSourceFormat

# Create CLEO III Navigation objects
prod sel NavigationProd
prod sel TrackDeliveryProd

# Find Pi0's and other decays to photon
prod sel PhotonDecaysProd

# Use "simple" electron and muon ID
prod sel SimpleMuonIdProd
prod sel SimpleElectronIdProd

# Load Splitoff producer
prod sel SplitoffProd
param SplitoffProd splitoffFilePath $env(C3_DATA)

```

The use inside your analysis code is identical in either case: you're provided with a list of approved `NavShowers` (with usage tag `"SplitoffApproved"`) to use as you see fit.

B.6 More Information from `SplitoffInfo`

More information about the details of the splitoff decision for each shower is available in the `SplitoffInfo` object created for each shower in the event. See the header file of this object for details, but the basic idea is that the “approved” `NavShower` list is constructed from only one of the bits of information stored in the `SplitoffInfo` object for a shower: whether it is approved for “use.” The object itself also stores the result of the neural net evaluation, the most likely primary or parent shower to which the shower is matched, the angular separation from this parent, and other splitoff identification details.

Some of the more useful methods are briefly described below.

- `DABoolean use() const`
When true, indicates this shower passed all splitoff rejection cuts and has been approved for use in neutrino reconstruction. This combines evaluation of the neural net appropriate for this shower's energy and location, along with energy, track-matching, and other quality requirements.
- `DABoolean isolated() const`
When false, indicates that this shower has an identified “parent” nearby from

which it is a possible splitoff. A parent shower generally lies within a small angular separation in ϕ and θ . If this function returns true, then no such shower has been found. **Note:** The initial value defaults to `true` *before* any parents are found. Thus a track-matched shower will generally be marked as “isolated” since no attempts were made to identify nearby parents!

- `DABoolean failedMinEnergyCut() const`
The first of several functions that indicate the reasons a shower may have been rejected for use. When true, this one means that the shower did not pass a basic minimum energy requirement of 25 MeV. (This energy cut is now a Parameter of the Producer and can be changed—at some risk.)
- `DABoolean badShower() const`
When true, indicates that the shower was flagged “bad” at reconstruction time, typically because the central crystal is known to have had some sort of (historical) problem. The value is filled identically with the one retrieved from `CcShowerAttributes::status()`.³
- `DABoolean matchedToTrack() const`
When true, indicates this shower has at least one track matched to it, based on the standard track-shower matching lists. Identically opposite to `NavShower::noTrackMatch()`. (Note, however, that if, in the future⁴, attempts are made to “reclaim” type 2-matched showers as genuine neutrals, an initial type 2 match may be cleared. At that time, this function will record only whether the shower is type 1-matched to some track.)
- `DABoolean shadowMatched() const`
Do not use! When true, indicates that the looser shadow-matching algorithm has matched this shower to a nearby parent. This algorithm is not used in `suez/CLEO III`, so this will never be set true, even for `CLEO II`.
- `DABoolean failedNeuralNet() const`
When true, indicates that this shower failed the neural net cut, meaning that the neural net has identified this shower as a likely splitoff based on its position, orientation, energy distribution, shape, and similar variables.
- `double netValue() const`
The actual value resulting from the evaluation of the neural net. If there were any difficulties calculating the net inputs, the net output value is set outside the range $[-1, 1]$, typically a more negative number such as -4 or -5 .

³See the `CLEO II badsh` documentation in the common block definition file for details about this variable in `CLEO II` data.

⁴This won’t happen in your lifetime.

- **ShowerId parentShowerId() const**
The identifier of the shower that has been identified as a the probable parent for this shower, *i.e.* the shower from which this one is “split-off.” If this value is zero, the shower is called “isolated” and a different set of cuts and nets are used to evaluate the likelihood that it is a splitoff.
- **TrackId parentTrackId() const**
A shower is designated as “nonisolated” when it has a track-matched shower nearby. This function returns the identifier of the track matched to that nearby shower. The function returns 0 (zero) if the shower is isolated, meaning there is no nearby matched shower.
- **double parentShowerSepAngle() const**
The space angle in steradians between this shower and its parent. The default value is 999 and is only changed from this for nonisolated showers.
- **double parentShowerSepTheta() const**
The separation between this shower and its parent in the ϕ -direction, in degrees. The default value is 999 and is only changed from this for nonisolated showers.
- **double parentShowerSepPhi() const**
The separation between this shower and its parent in the θ -direction, in degrees. The default value is 999 and is only changed from this for nonisolated showers.

B.7 Algorithm Overview

The heart of the splitoff identification algorithm is a set of finely-tuned and fairly sophisticated multi-layer neural nets. Hence the bulk of the code is simply calculation of the input variables to the net and then the handling of the net output. Certain special cases arise, of course, when some net inputs cannot be calculated, *e.g.* not enough near neighbors to form a clear pattern of energy deposition, or too close to the edges of the endcap to have clearly-defined or unique neighbors.

Only one particular net is evaluated for any particular shower, though there are many nets in the code (26 to be exact). The idea behind this design was to “help out” the net a bit, by having each focus on showers within a small range of energy. In addition, by not using shower energy as an input variable, one can guarantee that to first order the nets won’t be biased in energy. There are also separate nets for barrel and endcap showers, in recognition of the huge differences between barrel and endcap geometry. Rather than taxing a single net by asking it to handle but also be insensitive to these differences, it was deemed wiser to simply build and train separate neural nets for each class of showers. The final

complication is that showers are either *isolated* or *nonisolated*, which again doubles the number of nets.

“Nonisolated” showers are those for which another shower is near enough (20° in theta, 25° in phi) to be considered a likely parent. For these, the nets essentially assess whether the daughter shower “points” back to the parent shower. For the other class of unmatched showers (“isolated”), the nets determine whether the showers “point back” to the origin, *i.e.* are consistent with being (prompt) photons.

B.7.1 Neural Nets

Warning: The neural nets were originally (and only) trained on CLEO II data. This means they are *not* optimized for the different geometry and energy resolution of CLEO III. Nonetheless, we expect them to work “reasonably well” when applied blindly to CLEO III.

A neural net is really just a trainable decision-making algorithm: a set of input variables is fed in one side of a black box, some magic happens internally, and an output appears on the other side reflecting to what extent the combined variables have some signal characteristic. Typically, the inputs lie in the range $[-1, 1]$, and the output almost always has this range. The internals of the magic box are often thought of as layers of nodes or “neurons” that transmit “signals” to each other, but in practice, it is usually implemented with some simple matrix algebra. Each node accepts some number of inputs from the layer above, applies a weight to each, adds a possible bias term, and then funnels this result through a “response function” that maps the node’s output back into $[-1, 1]$ in a nice fashion. Fig B.4 below shows a typical layout of a neural net of two layers, with four inputs. Each layer of a net can have an unrestricted number of nodes, and every layer is always maximally cross-linked to the layer above and below it. Note that for coding purposes, the output node can be viewed as simply a terminal layer of one node.

The succinct description of the output z_i of the i -th node in some layer as a function of the inputs x_j is shown below. The weights for the inputs are the w_{ij} and the bias term for this node is b_i . The function f is the response function, typically something nice like $\tanh(x)$ that can accept any input and map it monotonically onto $[-1, 1]$ with a steep transition through $x = 0$.

$$\begin{aligned} y_i &= \sum_j w_{ij} x_j + b_i \\ z_i &= f(y_i) \end{aligned}$$

The real utility of neural nets is in the way in which the “weights” for each input to a node are determined: instead of deriving them deductively, one actually

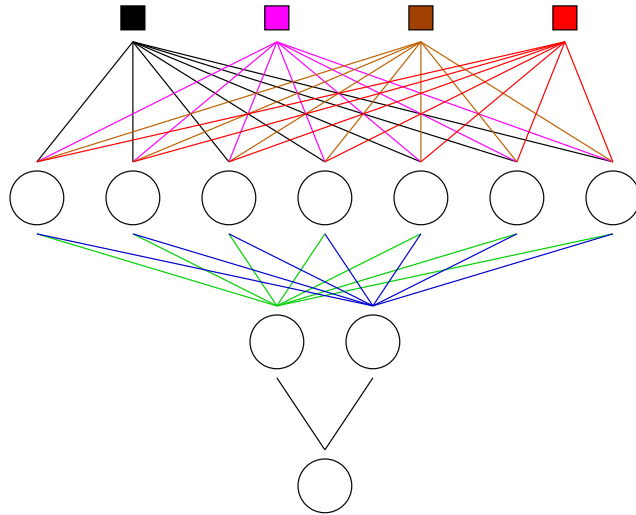


Figure B.4: A schematic of the architecture of a neural net. The boxes at the top represent input data, while the rows of circles represents layers of nodes or “neurons” that combine the inputs from the preceding layer to generate outputs that are then connected as shown to the inputs of the nodes in the next layer. The final layer of the net is a single output node.

“trains” the net on some sample data, and propagates back into the nodes an “error term” which modifies the weight at each input. (Typically one starts with a random distribution of weights across the net.) This feedback is the magical part—the weights tend to settle down after many iterations at close-to-optimal combinations that provide strong discrimination between signal and background for the sample data. After several such training sessions or “epochs” the net is mature enough to handle new data for which the right answer isn’t already known: the net becomes useful. To reconstitute the net at any time, one simply needs to know all the weights (and bias terms) in the net, for every input at every node. For a simple multilayer net as considered here, this is essentially a 3-D matrix, or a matrix for each layer. The `NeuralNet` library contains a basic implementation of such a neural net, and has methods that allow the user to set up a working net given the complete list of weights and information about the basic net configuration.

The details of the net configurations are given in Table B.1. Generally, each net has two layers, the first with 15 nodes, the second with 9. There are two nets for each energy bin, one for **barrel**, one for the **endcap**, for a total of **26 nets**. (The lower edge of each energy range is inclusive.)

Table B.1: The highest-energy net for nonisolated barrel showers omits the last net input $\log(B/G)$, and so has only 13 nodes in the first layer as a result.

Neural Nets for Nonisolated Showers		
Energy Range (MeV)	Number of Inputs	
	Barrel Net	Endcap Net
< 50	8	8
50 - 75	8	8
75 - 100	8	8
100 - 200	8	8
200 - 400	8	8
400 - 600	8	8
> 600	8	7

Neural Nets for Isolated Showers		
Energy Range (MeV)	Number of Inputs	
	Barrel Net	Endcap Net
< 100	6	7
100 - 200	6	7
200 - 300	6	7
300 - 400	6	7
400 - 600	6	7
> 600	6	7

B.7.2 Inputs

The inputs to the neural nets are discussed here. In summary, the position and orientation of the daughter shower relative to the parent shower are included, along with the shower's $E9/E25$ value and the pattern of energy deposition within the 3×3 block around the central crystal. Together, these inputs steer the neural net output to a value between (-1) for photons and $(+1)$ for splitoffs.

Inputs to the Nonisolated Shower Nets

1. $0 < \alpha < 45^\circ$ indicating “o'clock position” of parent shower around this candidate shower, folded into 45°
2. Separation angle between candidate and parent as measured from origin
3. Row-sum ratio for row of crystals closer to parent shower, in direction closest to being perpendicular to line of flight between the two

4. Row-sum ratio for row of crystals farther from parent shower, in perpendicular direction
5. Row-sum ratio for row of crystals closer to parent shower, in orthogonal direction to above
6. Row sum ratio for row of crystals farther from parent shower, in orthogonal direction
7. $X925 = E9/E25 \div CUT1$
8. $\log(B/G)$, where (B/G) is the “Bad/Good” ratio for the shower looked in up in a table based on the row-sum ratios

Inputs to the Isolated Barrel Shower Nets

1. $\log(B/G)$
2. $X925$
3. ϕ -sum ratio for row of crystals that has larger row-sum
4. ϕ -sum ratio for row of crystals that has smaller row-sum
5. θ -sum ratio for row of crystals closer to $z = 0$
6. θ -sum ratio for row of crystals farther from $z = 0$

Inputs to the Isolated Endcap Shower Nets

1. $0 < \alpha < 45^\circ$ indicating “o’clock position” of parent shower around this candidate shower, folded into 45°
2. $\log(B/G)$
3. $X925$
4. Row-sum ratio for row of crystals that is closer to origin, in direction perpendicular to hypothetical “line of flight” from $z = 0$ axis
5. Row-sum ratio for row of crystals that is farther from origin, in direction perpendicular to line of flight from $z = 0$ axis
6. Row-sum ratio for row of crystals that is closer to origin, in direction parallel to line of flight
7. Row-sum ratio for row of crystals that is farther from origin, in direction parallel to line of flight

Fig B.5 should help clarify the row- and column-sum ratios for a general non-isolated shower. The red crystal is the center of the candidate shower, with its eight nearest-neighbors indicated in light blue. The parent shower is off to the right, and the row-sums are the six sums of three crystals, each formed as shown. Given the line of flight from the parent shower to the splitoff candidate, one would expect the ratio of Sum 3 to Sum 1 to be most telling—it should be big, indicating that the bulk of the energy is on the side closer to the parent shower, because that’s where the energy came from. One expects the orthogonal sums (Sums A-C) to be weaker indicators of this possible parentage. The neural net is given just the *ratios* of the outer sums to the total, *e.g.* $\text{Sum 1}/(\text{Sum 1}+\text{Sum 2}+\text{Sum 3})$ and $\text{Sum 3}/(\text{Sum 1}+\text{Sum 2}+\text{Sum 3})$. The row-sums that are the best possible “pointers” back to the parent shower are always handed to the net first. (See input lists above.)

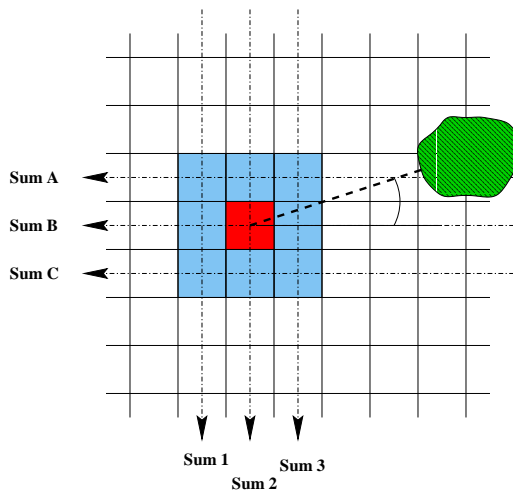


Figure B.5: A description of how the row- and column- sums are computed for a shower. The line of flight between the parent and daughter shower is used to determine in which direction (horizontal or vertical in this example) the sums will be most useful in determining whether the shower energy distribution is concentrated on the side of the daughter shower “nearer” to the parent.

For isolated showers, simply substitute the origin for the parent shower in the discussion above, and the treatment is fairly similar. In the barrel, the ϕ row-sums are all equidistant from the origin, so one simply chooses the order based on the larger of the two outer sums.

B.7.3 Cuts

A cut is applied to the neural net output to conclude whether a shower is either a splitoff or a genuine photon.

For nonisolated showers, the value of this cut varies with the shower energy (with a different granularity than that used for the nets), the “quality” of the best π^0 combination of which the shower is a part, the parent-daughter separation angle, the species of the parent track that is matched to the primary/parent shower, and the shower location (endcap/barrel). In the case that the net evaluation fails, an alternative cut is made on the shower energy, and is function of the same variables, except it is of course independent of shower energy.

For isolated showers, the value of this cut depends only on the shower energy, the “quality” of the best π^0 combination of which this shower is a part, and the shower location (endcap/barrel).

B.8 Code Overview

The top-level piece of code is of course the Proxy `SplitoffInfoProxy`, which creates and fills `SplitoffInfo` objects, and then places them in the Frame for others to use.

An outline of the steps taken by the Proxy in carrying out splitoff identification is enumerated below:

1. Extract the list of all showers in the event
2. Identify possible π^0 candidate pairings and classify how good each pair is in terms of distance from the nominal π^0 mass
3. Create a `SplitoffInfo` object for each shower
4. Weed out showers below a minimum energy cut, those that are track-matched (to approved tracks), and those that have a bad “status” flag
5. Loop over showers and identify track-matched showers as possible parents of unmatched ones
6. Loop over showers and create the appropriate `SplitoffNeuralNet` instance for each
 - (a) Calculate inputs for the neural net
 - (b) Evaluate the appropriate neural net for the energy, type of shower
 - (c) Apply cut to neural net result
7. Store neural net results in `SplitoffInfo` object for each shower
8. Put the finished `SplitoffInfo` objects in the frame.

Further details follow.

B.8.1 SplitoffProd

This code simply handles the registration of the Proxies, centralizes the various Parameters of the package, and conveys their values to the relevant pieces of code.

B.8.2 SplitoffInfo

This object contains most of the information used in reaching the splitoff decision and more details on the results. In some ways, it is the modern version of the old `splinf.inc` common block from the Driver package.⁵

B.8.3 SplitoffInfoProxy

User extraction of the “approved” shower list causes `suez` to internally trigger the extraction of the list of `SplitoffInfo` objects for the event. This Proxy provides those objects, one for each shower. As outlined above, it first determines which `NavShowers` are not matched to tracks and that pass a minimum energy cut. It then identifies the closest possible track-matched parent to each shower, and uses this parent candidate to make a splitoff likelihood assessment by evaluating a neural net.

B.8.4 SplitoffNeuralNet

The `SplitoffNeuralNet` object handles the details of calculating the inputs to the neural net for a single shower, evaluates the net, and then makes an optimized cut on the output to determine if this shower is a splitoff candidate. The results of these determinations are stored in the `SplitoffInfo` object described above. If the neural net evaluation fails for any reason, a few last-ditch rescue attempts are made.

There are two derived classes, one for isolated showers (`SplitoffNeuralNetIsol`), and one for nonisolated showers (`SplitoffNeuralNetNonisol`), which handle some of the ugly details of massaging shower variables into a form suitable for consumption by the neural nets. The base class essentially packages up the static information described later into a clean interface for use by the derived classes.

⁵If this is meaningless to you, thank the heavens above that the Dark Ages are finally over.

B.8.5 SplitoffApprover

This simple Selection Proxy builds a list of all `NavShowers` whose associated `SplitoffInfo` objects have `SplitoffInfo::use()` true. The resulting list can be extracted from the Frame with the usage tag "SplitoffApproved".

B.8.6 Pi0Protection

This is the bookkeeping object that determines the best π^0 pairing for each shower and saves the mass of this best combination for use in later cuts. It takes a list of the `NavPi0ToGGs` in an event.

B.9 Implementation Notes

This section contains notes about design decisions and implementation details that came up while authoring the new package.

B.9.1 Particle ID

As described above in Sec B.2.2, *Splitoff* needs basic particle ID information to help identify splitoff showers. The parent track species is not an input to the neural net, but it is used in looking up the cut to be applied to the net output. As might be expected, the cuts are most stringent for muons, somewhat relaxed for electrons, and loosest for hadrons, with energy dependence and other quantities folded in as well.

In order to make the package immediately usable to new consumers, “default” Producers have been provided to identify electron and muon tracks, but in a very simple, unsophisticated way. You should feel free to improve on and specialize your own copies of these for your own analysis, which perhaps already identifies electrons. A good starting point can be to simply modify the `ElectronSelector` object in `SimpleElectronIdProd` and incorporate your favorite EID criteria. Similarly for muons. Remember that the lists of muon and electron candidates should be disjoint!

B.9.2 “Row Sums”

Warning: Early versions of the Pass2 code did not properly calculate the so-called row-sums for CLEO III data, and meaningless numbers were stored in the Pass2 output. These sums form the basis for several of the neural net inputs, and

so the results of `SplitoffProd` cannot be guaranteed on CLEO III data processed with versions of Pass2 from before about Oct 2001.

In the process of coding up the algorithm in the new Producer, we identified a minor bug in the old `splitf` package, where it calculated the row and column sums for barrel showers near $\phi = 0$. The new code has this problem corrected (rather, it is corrected in `DriverCcBasicShowerAttsArg.cc`, where the sums are calculated for CLEO II data, and it is correct in the CLEO III Pass2 code). This is one reason (of many) that the new package will not agree exactly with the old one, even when running on the same CLEO II data. This particular difference gives rise to a new/old disagreement on typical hadronic showers at about the $\sim 0.01\text{--}0.05\%$ level.

B.9.3 Debug Output

Setting the report message level to `DEBUG` will produce volumes of output detailing the `splitoff` calculations for each shower, including the value of each of the neural net inputs, the value of the various net cuts, and for what reasons the attempt to evaluate the neural net may have failed.

B.9.4 CLEO II Details

Since `SplitoffProd` deals with high-level objects, the algorithm itself is insulated from the differences between CLEO II and CLEO III data. Changes were required when we first wrote the new package, however, to make sure that the CLEO III objects were properly filled with all of the data relevant to the `Splitoff` algorithm. This required two changes to `DriverCcBasicShowerAttsArg`, the piece of code in the `DriverDelivery` module responsible for constructing and filling the `CcBasicShowerAttsArg` objects for CLEO II roar data. In particular, we had to add the calculation of the so-called “row and columns sums” to the code, and define a “RingId” for each CLEO II crystal.

Calculation of the row and column sums for a CLEO II shower requires cell-by-cell energy information (available) as well as a prescription for how to add up the cells to get row- and column-sums of energy (not readily available). To eliminate the need for a call to `zfiles` to obtain nearest-neighbor lists for a crystal, we chose to instead store the lists directly in the code, since the lists for CLEO II are fixed and so need only be calculated once. The new `DriverCcShowerAttributesNbrInit.cc` file contains the ordered neighbor list for each crystal so that it is trivial to form these sums properly without any additional information.

Another change was the creation of an equivalent “RingId” for the CLEO II endcap, which allows `Splitoff` to determine whether a shower is too close to the

detector boundary (*e.g.* innermost or outermost ring of an endcap) to trust any pattern of energy distribution. A one-time map was constructed by hand, and a function `cleo3RingIdFromCleo2CellId(UInt16 cellId)` now maps CLEO II `CellId`'s onto "rings" in the calorimeter such that Rings 1, 10, 59, and 68 contain the inner east, outer east, outer west, and inner west endcap crystals that are too close to the edges of the endcap to have a set of at least seven clearly-defined neighbors. By using a properly-assigned `RingId`, we save `SplitoffProd` from having to look up geometry information repeatedly, and keep it CLEO II/III blind. The small amount of intelligence required for CLEO II access is built right into the code that fills the CLEO III shower objects from `roar`.

B.9.5 The *Splitoff* Data Files as Static Data

As mentioned earlier, the neural nets that form the key part of the `splitoff` identification algorithm were trained by L. Gibbons on carefully-controlled CLEO II data and Monte Carlo. The resulting net configurations and weights are stored in a set of files that are kept with the `SplitoffProd` package in the `SplitoffProd/Tables/` directory. There are also "lookup" tables for the so-called "bad/good" ratios that are calculated from the row and column sums for each shower. These files represent a significant amount of data, and were stored in formatted ASCII files, histogram files, and unformatted Fortran files in the old `splitf` package. We standardized all of them to ASCII text for simplicity, but could think of no better way to internalize the data than to simply read it all in at construction time. Since the `SplitoffNeuralNet` object has a lifetime matching that of a single shower, we decided to make all of this configuration and lookup data *static* so that it only need be read in and assembled once from the data files, and then accessed at no cost by any subsequent `SplitoffNeuralNet` instance.

The static initialization happens in the `init()` method of `SplitoffProd` itself, *i.e.* after the parameter specifying the path to the data files has been set. Similarly, the `terminate()` method takes care of the cleanup of the static data. Note that that if the `Parameter` value is changed after `suez` calls the `init()` method, nothing happens: the files, new or not, are not read in again. The intent is that the data be read in once for the entire duration of the job.

The files are:

- `isplit_net_weights.dat`

The configuration of the 12 nets for isolated showers, including the number of them, the energy range for each, and the node-by-node weights for each net.

- `nsplit_net_weights.dat`

The configuration of the 14 nets for nonisolated showers, including the num-

ber of them, the energy range for each, and the node-by-node weights for each net.

- **splrej.cuts**
The table of cut-values applied to the neural net output to identify a shower as a splitoff. These are applied to the neural net result, but vary with shower energy, π^0 quality, shower location, and parent species and angular separation for nonisolated showers.
- **e12.cuts**
The table of cuts applied when the neural net evaluation fails for nonisolated showers. These cuts are applied to the shower energy, but vary with shower location, π^0 quality, angular separation from parent, and parent species.
- **split2D.dat**
The lookup tables for “bad/good ratios” for nonisolated showers.
- **split1D.dat**
The lookup tables for “bad/good ratios” for nonisolated showers that have one vanishing row or column sum.
- **phit1d.dat, phit2d.dat**
The lookup tables for “bad/good ratios” for isolated barrel showers for ϕ row sums.
- **thet1d.dat, thet2d.dat**
The lookup tables for “bad/good ratios” for isolated barrel showers for θ row sums.
- **bstt1d.dat, bstt2d.dat**
The lookup tables for “bad/good ratios” for isolated endcap showers for “best” row sums, the ones that provide greater discrimination in determining whether the shower “points back” to the origin.
- **sndt1d.dat, sndt2d.dat**
The lookup tables for “bad/good ratios” for isolated endcap showers for “second-best” row sums, the ones that provide worse discrimination in determining whether the shower “points back” to the origin.

B.9.6 Neural Nets

The `SplitoffNeuralNet` class uses the very simple `NeuralNet` library to implement its private internal nets. Note there are 6 (energy bins) \times 2 (barrel/endcap) = 12 neural nets for isolated showers, and $7 \times 2 = 14$ nets for nonisolated showers.

No provision has been made in the current package for re-training the neural nets.

B.10 Comparing Driver `splitf` and `suez SplitoffProd`

Part of the purpose in writing `SplitoffProd` was to allow `suez` users to employ `splitoff` rejection techniques while running over CLEO II data, where there already is a Fortran/Driver package providing the same functionality. This provides an ideal testing ground for testing the accuracy with which the original algorithm has been reproduced. Generically, we’ve found agreement at the 99.5 – 99.95% level. The few differences in whether a shower should be used or not are well understood, as described below. It has also been confirmed that the new implementation of the neural net gives results identical to the original one in every way, to arbitrary accuracy.

In a Monte Carlo sample of 5000 events, or 128921 showers, 56K were approved by the original algorithm. Subjecting the same sample to the new code, we found only 258 instances of disagreement.⁶ These differences are dominated (135 cases) by differences in PID used in the tests. The Driver job used MC truth while while the new algorithm employs “simple” reconstructed lepton ID as described above. The next biggest source of disagreement is simply crystals which are too close to the boundary of the endcap or barrel to be reasonably considered for `splitoff` rejection. The new code is more conservative and flags an additional 40 showers as hard-to-tell and so falls back on secondary methods for determining their `splitoff` character, to different result. The next largest reason for difference is a slight difference in the interpretation of the old common block variable `BADSH` which need not concern us here.

Fig B.6 is a comparison of the response of the neural net for the ~ 60 K showers for which the neural net was successfully evaluated. The results from `SplitoffProd` are overlaid in red and the neural net response as computed in `splitf` is underneath in black.

In short, the agreement is excellent, and the few cases of disagreement indicate no systematic problems or causes for concern. The new package is a completely suitable (and excellent) replacement for the old.

B.11 Updates/Revisions

- **2004.09.17**

Some small changes, extending the interface in simple ways:

- The `SplitoffInfo` object has been modified slightly: it now also stores the track id of the track matched to the nearby “parent” shower for non-isolated showers. The redundant inclusion of the `CcShowerAttributes`

⁶Note that for the purposes of this comparison, the $\phi = 0$ row-sum computation bug mentioned in Sec B.9.2 was repaired in the Fortran code.

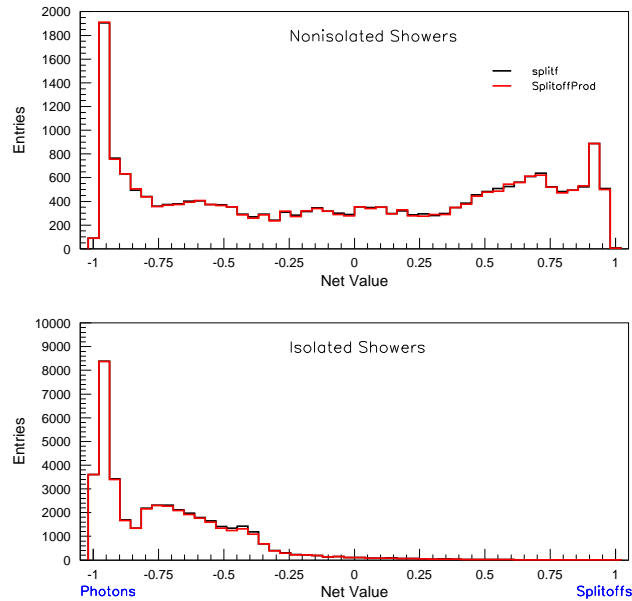


Figure B.6: A comparison of the neural net output between the old Fortran `splitf` and new suze `SplitoffProd` packages, based on a sample of 5000 $B\bar{B}$ Monte Carlo events. The plot shows the neural net response for showers for which the net inputs were all successfully computed. The `splitf` results are in black; the `SplitoffProd` results are overlaid in red.

object in this object has also been eliminated, simplifying and purifying the interface.

- The user now has the ability to specify the minimum energy a shower must have to be accepted. See Sec B.3 above.
- The Producer now prints out the run-time values of all Parameters at the start of the job (actually, in the `init()` method) so it is clear what values are being used.

For use in systematics evaluation, a new Parameter has been introduced to allow the user to turn on “smearing” of the neural net result before cuts are applied. See Sec B.3 above for a description of the Parameter, and Sec B.4 for a broader discussion of how to use this feature.

- **2002.05.31**

The `splitoff` data tables (net configuration, weights, net cuts, etc.) are now installed into the `C3_DATA` area of each software release as part of the normal build cycle. (New rules in the `Makefile` make this magic happen.) Thus users can now set the `splitoffFilesPath` parameter to `C3_DATA` as shown in Sec B.3 above and need no longer have a personal copy of the files.

- **2002.03.27**

Two features were added to the basic package:

- Added ability to select a subset of all tracks to be used for track-shower matching. The user can supply a shortened list of tracks “approved” for matching; *Splitoff* will ignore any matches made to tracks not in this list. See details in Sec B.2.3.
- Moved particle ID *out* of *Splitoff* to allow user flexibility in determining what’s an electron, what’s a muon, and what’s a hadron. See details in Sec B.2.2.

B.12 Some References

Some of the few written references on the old `splitf` package and the new `SplitoffProd`.

- **CBX 95-6**
Debut of neutrino reconstruction in the $B \rightarrow \pi/\rho\ell\nu$ analysis. Contains some of the early proof-of-principle and quality-checking plots for the original CLEO II `splitf` package.
- **S. E. Roberts Thesis (Rochester)**
Scott’s inclusive analysis also employed neutrino reconstruction, and includes a descriptive chapter on the splitoff-identification algorithm.
- **Neutrino Reconstruction Tools For Suez**, PTA Talk 12/07/2001
A few slides here describe the initial release of `SplitoffProd`, as presented to the B-Leptons PTA.
- **CLEO3 CVS: SplitoffProd**
The source code.
- **CLEO3 CVS: NeuralNet**
The source code for the library used to represent the neural nets in the `Splitoff` package.