Monte Carlo matching in the Belle II software

Yo Sato[1,*], *Sam* Cunliffe[2,**], *Frank* Meier[3,***], and *Anze* Zupanc[4]

[1]High Energy Accelerator Research Organization (KEK), Tsukuba, Japan.
[2]Deutsches Elektronen-Synchrotron (DESY), Hamburg, Germany.
[3]Duke University, Durham, USA.
[4]Jozef Stefan Institute and Sinergise LTD, Ljubljana, Slovenia.

Abstract. The Belle II experiment is an upgrade to the Belle experiment, and is located at the SuperKEKB facility in KEK, Tsukuba, Japan. The Belle II software is completely new and is used for everything from triggering data, generation of Monte Carlo events, tracking, clustering, to high-level analysis. One important feature is the matching between the combinations of reconstructed objects which form particle candidates and the underlying simulated particles from the event generators. This is used to study detector effects, analysis backgrounds, and efficiencies. This document describes the algorithm that is used by Belle II.

1 Introduction

The Belle II experiment [1, 2] at the SuperKEKB e^+e^- collider [3] is the successor to the Belle experiment [4]. The full software stack has been rewritten for Belle II: notably the tracking [5], clustering, and high-level analysis tools [6–8]. The Belle II software framework (basf2) [9] supports both C++ and python 3 code. The software is organised into *modules*[1], each of which perform a single task of data-processing within the event loop. Basf2 modules are configured in an ordered sequence, called a *path* which defines the processing of events in a job.

2 Monte-Carlo sample generation in the Belle II software

Belle II data are processed centrally within the collaboration. The output of this processing is a set of common post-reconstruction files[2] (containing data objects such as tracks, calorimeter clusters, particle identification information, etc).

Monte Carlo (MC) simulated data is reconstructed in the same way, with the obvious exception that events are generated and then passed through a GEANT4 [10] simulation [2, 11]. For *b*-physics analyses, the EvtGen [12] generator is typically used, and for tau physics, TAUOLA [13]. There are a variety of other generators available for low-multiplicity processes.

*e-mail: yosato@post.kek.jp
**e-mail: sam.cunliffe@desy.de
***e-mail: frank.meier@duke.edu

[1]An unfortunate nomenclature clash here: basf2 modules are distinct and different from a python 3 *module*.
[2]So-called mini data summary table (mDST) format files. Based on ROOT.

In the post-reconstruction files from simulated events, the underlying generator-level particles are also stored alongside the reconstructed tracks, clusters, and other information. The generator-level particles are described by a `Belle2::MCParticle` data object.

These post-reconstruction files (or subsets thereof) are processed again by the analyst who will have specific analysis requirements for her physics channel.

3 High-level analysis at Belle II

In high-level analysis jobs, the tracks and clusters (or combinations thereof) are interpreted as candidate physics particles (`Belle2::Particles`). For example, a track associated to a calorimeter cluster consistent with a minimum-ionising particle, and with hits in the muon system would be a muon *candidate* particle. No specific selection criteria are necessarily imposed at this stage, any and all tracks may be interpreted as "muon" particles. To give another example, two particles under a kaon and pion hypothesis (using, say, particle identification information) can be combined to make a *composite* particle candidate for a $K^{*0} \to K\pi$ decay.

It is at this stage, and to these particle candidates, that analysis-specific algorithms (such as kinematic fitting, full-event interpretation [6], etc) are employed.

4 MC-matching

The procedure of identifying the underlying generator-level particle that is responsible for a reconstructed candidate is *MC-matching* and is important for understanding reconstruction effects and backgrounds. This is the association of a particle to an MC-particle in Belle II nomenclature.

The MC-matching of tracks is described in Ref. [5]. In brief: it is a requirement on the purity of the hits used in the track that are due to a single generated particle. Similarly, a cluster is MC-matched if a certain fraction of its energy is due to the generated particle. Track-based and cluster-based particles (final-state particles) therefore simply inherit the match of the underlying reconstructed object, with preference to the track. For example, if the track of the muon candidate particle discussed earlier is matched to a generator-level muon, then the particle will inherit this match regardless of the match of the associated calorimeter cluster. Photons, K^0_L, and neutrons take their MC match from that of the associated cluster.

For composite particles Belle II employs a two stage process. An algorithm to find an MC-match is followed by an *evaluation* step to categorise incorrectly reconstructed candidates.

5 The algorithm for composite particles

The goal of this MC-matching algorithm is to establish the relation between the reconstructed composite particle and the corresponding MC-particle. A flowchart of the algorithm is shown in Fig. 1.

Let us take the following decay chain as an example: $\Upsilon(4S) \to B^0\overline{B}^0$ followed by $B^0 \to K^{*0}(\to K^+\pi^-)e^+e^-$. Let us assume that the B^0 candidate is correctly reconstructed from final-state particles (K^+, π^-, e^+, and e^-), which have already been matched with the corresponding MC-particles. In this case, one wants to assign the MC-particle B^0 as the match to the candidate. The overall algorithm process is as follows:

- Check that every daughter (K^{*0}, e^+, e^-) of the initial particle (B^0) is a composite particle or a final-state particle.

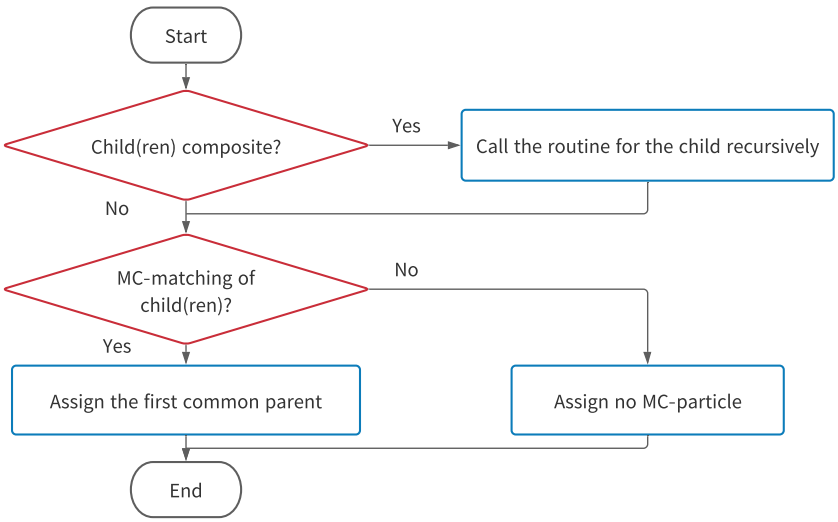

Figure 1. The algorithm flowchart of the MC-matching for a composite particle.

- If a daughter is a composite particle, such as the K^{*0} in the example, call the algorithm routine for that daughter recursively.
- Check all daughters of the initial particle for any that have already been matched.
- If all daughters have been matched, assign the most recent common ancestor from all MC-particles which are matched to the daughters. Hereafter, the most recent mutual ancestor is called *the first common mother*. In the example, the first common mother of the (K^{*0}, e^+, e^-) is the B^0. Thus, the MC-matching of B^0 is correctly established. The MC-matching of the K^{*0} will also be correct in the example, since the first common mother of (K^+, π^-) is the K^{*0}.
- If any daughter has not been matched, and no MC-particle is matched either: the algorithm is running on data, a final-state particle has been created from beam-background hits, or a particle is otherwise wrongly reconstructed.

It is also possible that a B^0 candidate may have been reconstructed partially from correctly matched particles (say, K^+, π^-, e^+) and a wrong particle (e^-_{other}) which could be a decay product from, for instance, the other B. In this case, the first common mother of (K^+, π^-, e^+, e^-_{other}) is the $\Upsilon(4S)$. So the B^0 candidate will be matched to the $\Upsilon(4S)$.

If there is only one daughter in the "composite" particle, then the particle is assigned a match to the mother of the MC-particle matched to this daughter. This is a fringe case for, for example, semileptonic decays such as $B^+ \to \tau^+\nu_\tau$. Assuming that the tau is correctly reconstructed in this example, then the particle has only one reconstructed mother. It is also sometimes desirable for users to "combine" a single particle with nothing to create a "mother". This is helpful for bookkeeping, and labelling purposes.

6 Evaluation algorithm for composite particles

It is useful to categorise failure cases of the reconstruction. For this, a so-called *evaluation algorithm* is employed as a second step to the MC-matching.

The MC-matching of final-state particles can be evaluated trivially. On the other hand, evaluating the matching of composite particles must consider not only the MC-matching of the given particle but also that of daughters and particles which may be missed or erroneously added. Moreover, there must be the option to take into account the requirements of important physics analysis such as inclusive analyses (e.g. $B \to X\gamma$). The MC-matching evaluation provides several error flags, that each analyst can choose to accept or not for her own analysis use case.

The evaluation for composite particles is divided into two major parts: processes with existing particles and processes with missing particles. Figure 2 shows the algorithm flowchart of the evaluation for existing particles. The algorithm is as follows:

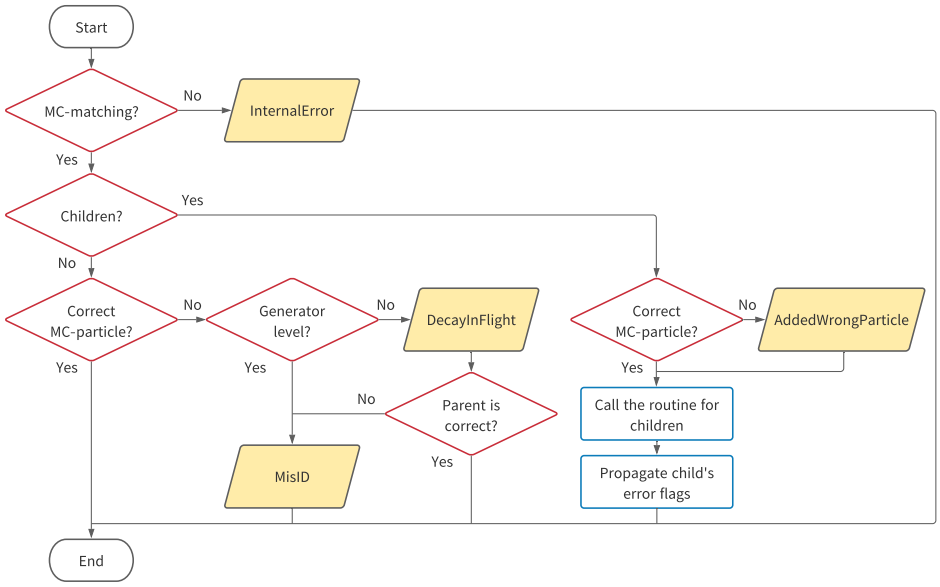

Figure 2. The algorithm flowchart of the evaluation for existing particles

- First, check if the given particle has a match. If not, an `InternalError` flag is added indicating a problem in the matching.
- Check if the given particle has daughters.
- If the particle does not have any daughters, check if the MC-matching between the given particle and the MC-particle is correct.
  - If the MC-matching is correct, the routine ends.
  - If the MC-matching is not correct, check if the MC-particle related to the given particle is created in the GEANT4 detector simulation or in the generator.
    - If the MC-particle is simulated at generator level, then the `MisID` flag is added since the particle identification was not correct.
    - If the MC-particle was created at detector level, then the `DecayInFlight` flag is added since the particle must have decayed in the detector. And if the mother MC-particle of the MC-particle is also different from the given particle type, the `MisID` flag is also added.

- If the particle has daughters: check if the MC-matching is correct for each daughter. If not, the `AddedWrongParticle` flag is added.
  - The same algorithmic routine is then called recursively for all daughters.
  - The error flags of daughters are propagated to the parent particle.

Figure 3 shows the algorithm flowchart of the evaluation for missing particles. The algo-

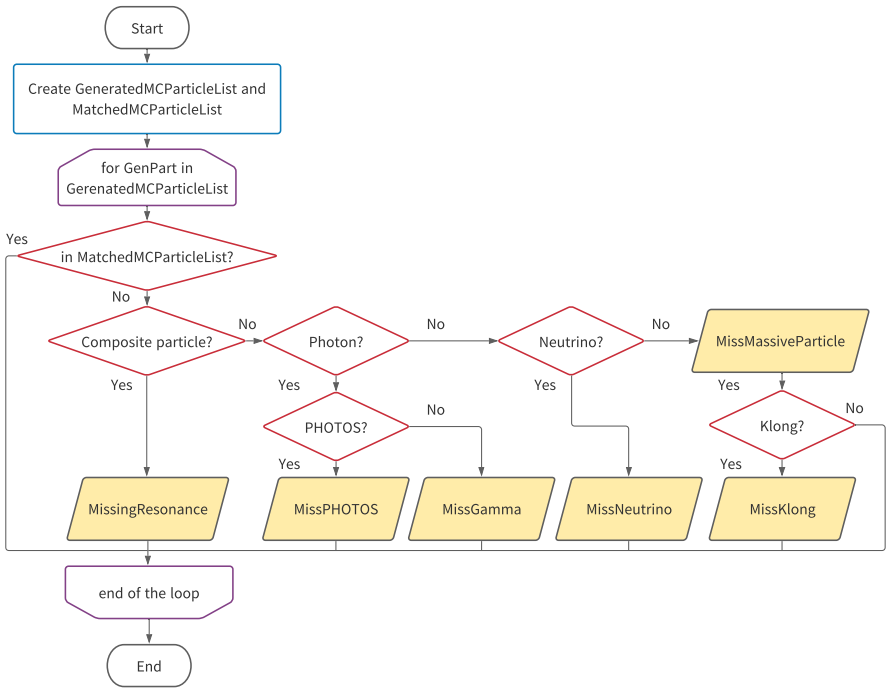

Figure 3. The algorithm flowchart of the evaluation for missing particles

rithm is as follows:

- Create two lists of MC-particles: *GeneratedMCParticleList* which has all MC-particle daughters of the MC-particle which is matched to the given particle, and *MatchedMC-ParticleList* which has all MC-matched particles of the reconstructed decay chain.
- Check if each generated MC-particle of the *GeneratedMCParticleList* is included in the *MatchedMCParticleList*. If a generated MC-particle is not included, then a MC-particle was missed from the reconstruction. The cause of the missing particle is identified as follows:
  - If the MC-particle is a composite particle, then the `MissingResonance` flag is added.
  - If the MC-particle is a photon, first check if it is produced with PHOTOS [14] which simulates QED radiative corrections. In this case, the `MissPHOTOS` flag is added. If the MC-particle is produced with another generator such as EvtGen, then the `MissGamma` flag is added.
  - If the MC-particle is a neutrino, then the `MissNeutrino` flag is added.

– Otherwise, the `MissMassiveParticle` flag is added since the other particle was not a photon or neutrino.

– If the MC-particle is K_L^0, the special `MissKlong` flag is added.

7 User interface

The matching of composite particle candidates can be computationally expensive if there are large combinatorics. Composite particles are therefore not matched or evaluated by default. The `MCMatcherParticles` module must be added to the basf2 processing path. This module runs both algorithms. A very brief script to run our example analysis from Section 5, is:

```
import basf2
path = basf2.Path()  # instantiate a basf2 path
path.add_module("RootInput", "/path/to/a/file.root")
path.add_module("ParticleLoader", ["e+", "pi+", "K+"])
path.add_module("ParticleCombiner", "K*0 -> K+ pi-")
path.add_module("ParticleCombiner", "B0 -> K*0 e+ e-")
path.add_module("MCMatcherParticles", "B0")  # matching module added
path.add_module("VariablesToHDF5", "B0", ["InvM", "isSignal"])
basf2.process(path)  # execute the processing event loop
```

In this script, tracks are interpreted as particle candidates under three hypotheses: $e^{\pm}$, $\pi^{\pm}$, $K^{\pm}$ (`ParticleLoader`). They are then uniquely combined to create candidate $B^0 \to K^{*0}e^+e^-$ decays comprising of groups of four tracks (`ParticleCombiner`). The penultimate module runs the MC-matching algorithm over all candidates. The final module writes out each candidate's invariant mass and a boolean `isSignal` quantity. The latter could be used as a target for some machine learning classifier, for example.

The user can control the behaviour of the algorithm and options to accept missing resonances, etc. with the use of syntax in the *decay string*. To give some examples:

- `"@Xsd -> K+ pi-"` specific resonances (marked by `@`) that are correctly reconstructed do not need to be individually specified and are labelled X_{sd}.
- `"B0 =exact=> K*0 e+ e-"` requires an exact match of every decay particle.
- `"B0 -> D*- tau+ ?nu"` a missing neutrino is acceptable.

8 Summary

In summary, MC-matching is the association of particle candidates to the underlying generator-level particle. For track-based and cluster-based particle candidates, the match is inherited from the track or cluster (track preferred). For composite particle candidates, the Belle II algorithm attempts to find the *first common mother* of daughters. The user interface makes use of the *decay string* to configure the matching algorithm in an intuitive way.

References

[1] T. Abe et al. (2010), `1011.0352`
[2] W. Altmannshofer et al., PTEP **2019**, 123C01 (2019), [erratum: PTEP2020,no.2,029201(2020)], `1808.10567`
[3] K. Akai, K. Furukawa, H. Koiso, Nucl. Instrum. Meth. **A907**, 188 (2018), `1809.01958`

[4] J. Brodzicka et al., PTEP **2012**, 04D001 (2012), 1212.5342
[5] V. Bertacchi et al., Comput. Phys. Commun. **259**, 107610 (2021), 2003.12466
[6] T. Keck et al., Comput. Softw. Big Sci. **3**, 6 (2019), 1807.08680
[7] J.F. Krohn et al., Nucl. Instrum. Meth. A **976**, 164269 (2020), 1901.11198
[8] F. Abudinén et al. (2020), 2008.02707
[9] T. Kuhr, C. Pulvermacher, M. Ritter, T. Hauth, N. Braun, Comput. Softw. Big Sci. **3**, 1 (2019), 1809.04299
[10] S. Agostinelli et al. (GEANT4), Nucl.Instrum.Meth. **A506**, 250 (2003)
[11] D.Y. Kim et al., J. Phys. Conf. Ser. **898**, 042043 (2017)
[12] D.J. Lange, Nucl. Instrum. Meth. **A462**, 152 (2001)
[13] S. Jadach, J.H. Kuhn, Z. Was, Comput. Phys. Commun. **64**, 275 (1990)
[14] P. Golonka, Z. Was, Eur. Phys. J. C **45**, 97 (2006), hep-ph/0506026