Integration of the RoadSearch Tracking Algorithm into the Porting of the ORCA Track Reconstruction into CMSSW

Kevin Burkett, Oliver Gutsche

Fermilab

Steve Wagner

University of Colorado

November 7, 2005

1 Introduction

In this note we describe a proposal for the implementation of the Road Search (RS) tracking algorithm in CMSSW. We first review the general method that is used by the RS algorithm. Then we describe an initial implementation in CMSSW, where we try to stick as close as possible to the current RS implementation in ORCA. In the description of this implementation, we present a modular breakdown of the steps involved in the algorithm, to allow the interchange of individual steps with the corresponding steps of the CombinatorialTrackFinder (CTF). A proposal for the modularity of all track reconstruction algorithms was at the Software Meeting on October 24, 2005. (See "Report on Reconstruction" by Tommaso Boccali [1].) The proposed structure is shown in Figure 1, and the proposal for the RS implementation presented here fits into this model. We consider the reusability of existing track reconstruction functionality being ported from ORCA to CMSSW.

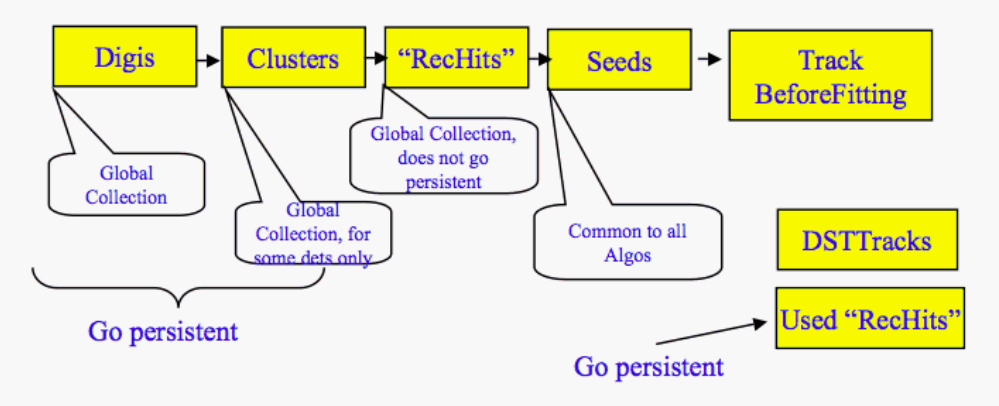

Figure 1: Proposed modular flow of track reconstruction algorithms.

2 Road Search Algorithm Overview

In the RS algorithm hits are collected along helical paths in the tracker, referred to as roads, and then fit to make tracks. In the current implementation of the RS algorithm in ORCA, the tracking detectors are grouped in rings of detectors in ϕ. The tracker can then be projected in the $r-z$ plane, where particles follow linear trajectories. The roads are pre-defined sets of expected detector hits in $r-z$. These roads all have specified inner and outer starting rings that are used to define the road. Note that while future RS implementations may include information from detectors other than the tracker, only hits in the tracker are used at this time.

The input to the RS is a hit collection, where the hits contain local coordinates and errors. The RS loops on pairs of hits, one each from the inner and outer starting rings, and applies a loose $\Delta\phi$ cut. If the hits pass the $\Delta\phi$ criteria, a trial helix is calculated, including the initial hit pair and the beam spot. The trial helix is projected in either $r-\phi$ or $z-\phi$, depending on the starting rings. Hits are then collected within a window around this helix in the other rings of the pre-defined road. This collection of hits is referred to as a "cloud". After looping over all input roads, the clouds are compared to each other to merge similar or redundant clouds. From the collection of all clouds, we attempt to fit tracks. In ORCA, the TrajectoryBuilder framework was used for this step. The port of the CTF to CMSSW will provide this functionality. The final output is then a collection of DSTTracks, fit in the same manner as with the CombinatorialTrackFinder.

3 Implementation of Road Search in CMSSW

In CMSSW the starting point of the RoadSearch is the collection of tracker hits in the event. Starting from Clusters, the hits are lightweight objects that give the hit position in local coordinates along with its error, and are referred to as "RecHits" in Figure 1. There was no collection of such objects in ORCA. (Note that we have written "RecHits"with quotation marks to denote the fact that these are not the same "RecHits" that existed in ORCA.) From the collection of "RecHits" we create Seeds for the Road Search. The Seeds contain the pair of hits from the starting Rings that passed the $\Delta\phi$ cut. Using the Roads defined by the Rings of the "RecHits" from the Seed, we collect "RecHits" on the other Rings of the Road to create Clouds. A subset of "RecHits" within the Cloud, consistent with a single track hypothesis, is selected and passed to a final track fit. This set of "RecHits" is referred to as the Used"RecHits" in Figure 1. The final track fit is the same as in ORCA, returning in the end, a DSTTrack.

Following the modular structure shown in Figure 1, the overall modular structure of the RS algorithm is shown in Figure 2. Note that we have explicitly shown that the creation of Tracks Before Fitting, a step that can be thought of more generally as Pattern Recognition, is really two separate steps of making Clouds and then Tracks Before Fitting in the RS algorithm. In the following sections we describe in greater detail the steps of the RS implementation, broken up into the structure of EDProducers.

3.1 "RecHit"Maker

Input: Clusters (DetId, vector<amplitude> , firststrip, barycenter)

Output: "RecHits" (DetId, LocalPoint, LocalError, *Cluster)

The input to the "RecHit"Maker is the collection of clusters in the event. Clusters contain the DetId of the associated detector, a vector of the amplitudes in the cluster, the strip number of the first strip in the cluster, and the barycenter of the cluster. Using the geometry the barycenter is converted to a position and error in local coordinates. At this stage the cluster is also corrected for the Lorentz angle, assuming a track of normal incidence. The "RecHit" includes a pointer to the Cluster from which it was created.

The creation of the "RecHit" should include the handling of glued detectors, combining the 1D clusters on a matched pair of detectors into a single 2D DetHit, using the current

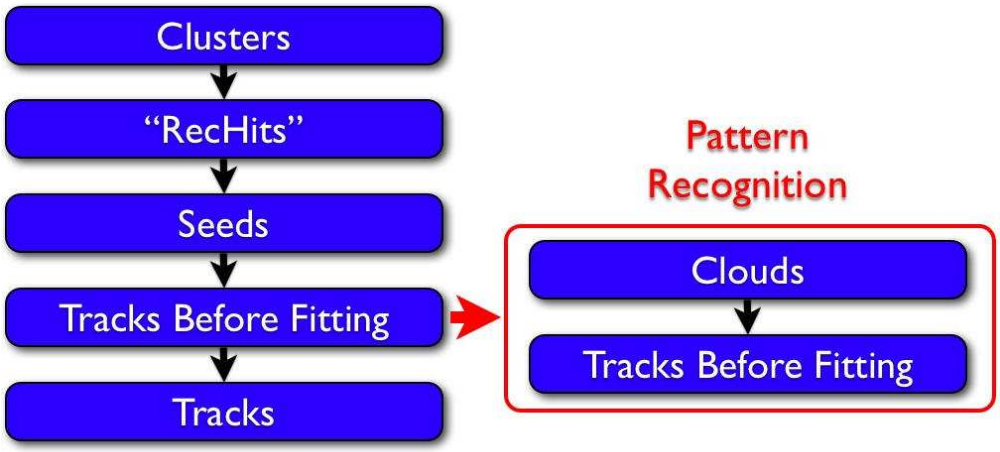

Figure 2: Modular flow of the DataFormats of the RoadSearch algorithm. The step of making Tracks Before Fitting has been split up to explicitly show the creation of Clouds in the RS algorithm. This is unique to the RS.

ORCA implementation. However, "RecHits" of different dimensions may be stored in separate collections.

3.2 SeedFinder

Input: "RecHit" Collection

Output: Seeds (trajectory, trajectory error, vector<*"RecHit">)

In the current RS implementation, roads are based on pre-defined sets of rings. Roads are seeded by inner and outer initial rings that serve as a starting point for the road. To create Seeds we loop over the <ring,ring> pairs of all pre-defined roads. A pair "RecHits" in the inner and outer starting rings, passing the $\Delta\phi$ cut form a seed. With the addition of the beamspot, a helix can be defined along with its errors. The Seed is a trajectory (five helix parameters) with errors, and includes a vector of pointers to the "RecHits" used to make the seed.

3.3 Implementation of Roads

The current implementation of the RS algorithm depends on predefined sets of Roads within the detector. Here, we describe the implementation of the Roads in CMSSW. The Roads will

be provided to the EDProducers by an ESProducer in analogy to the geometry service. The ESProducer will be capable of reading in the predefined Roads from a file or constructing them on-the-fly.

The Roads are selected groups of Rings in the detector. As we consider only the tracker in this document, a Ring corresponds to all sensors within a specified $r - z$ range summed over ϕ. In CMSSW a Ring is a vector of detector id's, following the detector id definition [2]. As an example, the TIB detector id is defined by:

- layer
- forward/backward bit
- external/internal bit
- string
- module
- stereo bit

where the string number represents the segmentation in ϕ. A TIB Ring contains all detector id's of the TIB with a defined combination of all parameters except the string parameter. All Rings of the tracker are constructed by iterating over the detector id parameters and using the dedicated detector id classes provided by the framework.

The Roads then point to a specific set of Rings and are indexed by their inner and outer Seed Rings. Having chosen an inner and outer Seed Ring pair, a linear extrapolation is performed in $r - z$ from the Beamspot (including a 3σ spread in z) through the minimal and maximal extents in r and z of the Seed Rings and all Rings intersected are filled in the Road container. The container is a multimap whose key is the pair of inner and outer Seed Rings which index the vector of all Rings of the Road. The Roads ESProducer depends heavily on the correct implementation of the geometry including the detector id schema.

3.4 Pattern Recognition

3.4.1 CloudMaker

Input: Seed Collection

Output: Clouds (vector<*DetHit>, *Seed)

The CloudMaker uses the DetIDs of the two "RecHits" in the Seed to lookup the <ring,ring> pair in the multimap that contains the pre-defined roads. The CloudMaker then loops over the set of roads for the input pair. For each intermediate ring in the road we loop over the "RecHits" in that ring. If the hits fall within a window in ϕ around the Seed, they are added to the Cloud. The Cloud contains a vector of pointers to "RecHits" and a pointer to the Seed used in the creation of the Cloud. A cut on the minimum number of "RecHits" in the Cloud is made to decrease the number of Clouds coming from random combinations of "RecHits". After completion of the loop over all roads, Clouds are compared to merge similar or redundant Clouds in the output collection

3.4.2 TrackFinder

Input: Cloud Collection

Output: Used"RecHits" (vector<*DetHit>, *Cloud)

A subset of "RecHits" within the Cloud, consistent with a single track hypothesis, is selected. The vector of "RecHits" is saved, along with a pointer to the parent Cloud, in the Used"RecHits".

3.5 TrackFitter

Input: Used"RecHits"

Output: DSTTracks

In the final step of tracking, the TrackFitter loops over the vectors of "RecHits" in the Used"RecHits" and performs the final track fit. The same TrackFitter is used by both the RS and CTF.

3.6 Summary

The modular structure of the RS algorithm is summarized in Figure 3, along with the DataFormats and EDProducers at each step. The details of the DataFormats and EDProducers have been given in the preceding sections. Note that the implementation presented here shows the minimal requirements of the RS for the DataFormats and EDProducers. The addition of data members and implementation features needed by the CTF should be straightforward.

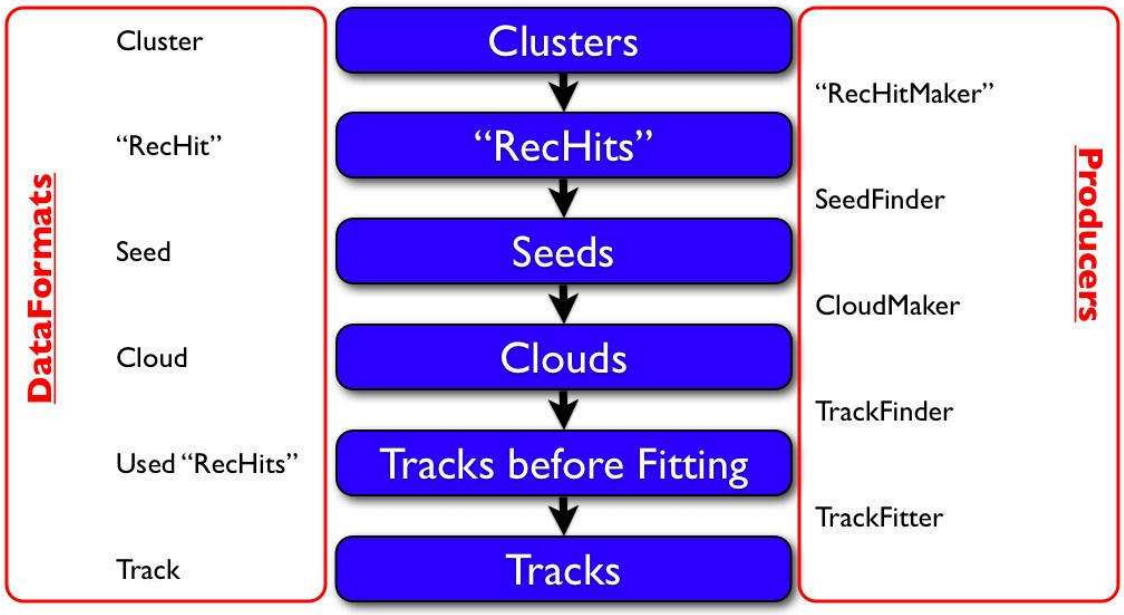

Figure 3: Modular flow of the RoadSearch algorithm, with the DataFormats and EDProducers for each step.

References

[1] http://agenda.cern.ch/fullAgenda.php?ida=a056489

[2] N. Ampane *et al*, "Models of Data for Tracking Detectors". Draft CMS Note.