AMBIGUITY RESOLVER

MIRIAM DIAMOND

JUNE 5 2016

github issue 65

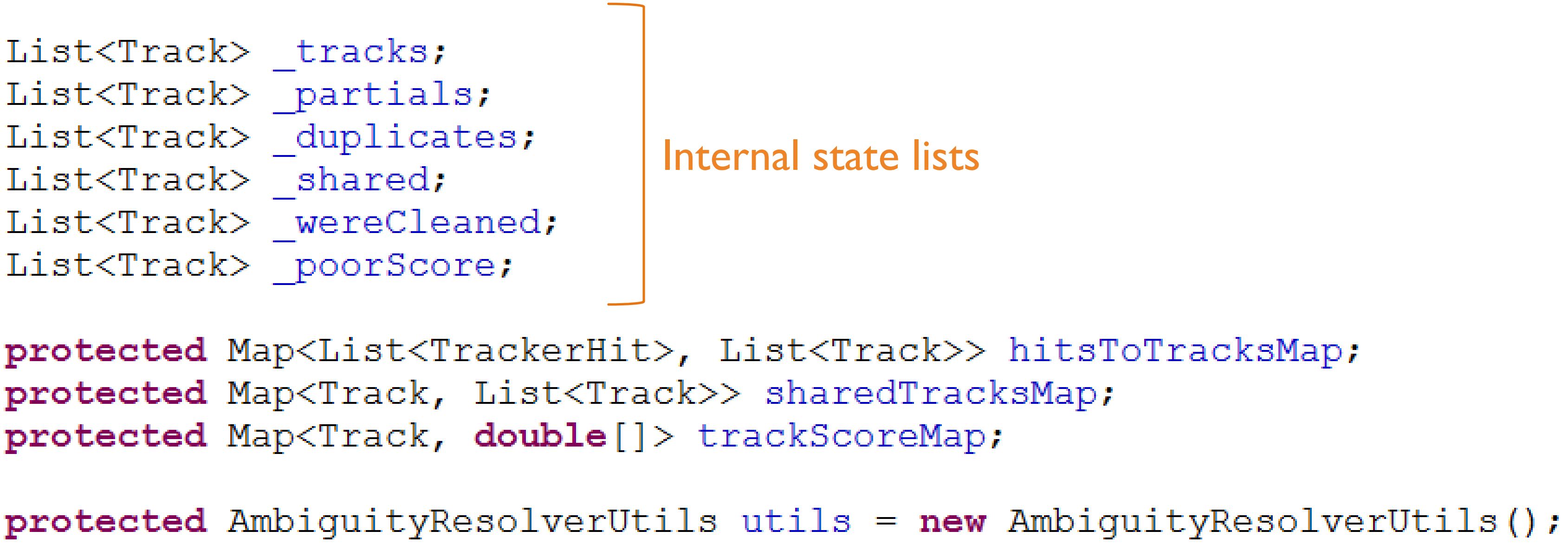
Definition:

- Chooses between multiple tracks, only one of which is likely to be real
- May impose track quality requirements
- May perform "track cleaning" (i.e. discarding outlier hits)

CURRENT SITUATION IN HPS SOFTWARE

- hps-tracking ▶ src/main/java ▶ org.hps.recon.tracking ▶ MergeTrackCollections driver runs after GBL
- Performs only most basic ambiguity resolving tasks: removal of duplicate and partial tracks
  - Duplicates: were formed using different strategies
  - “Partial”: entire track is an exact subset of a longer track in the collection
- Further ambiguity resolving tasks performed ad hoc by individual analyses

AMBIGUITY RESOLVER ABSTRACT CLASS



```
List<Track> _tracks;
List<Track> _partials;
List<Track> _duplicates;
List<Track> _shared;
List<Track> _wereCleaned;
List<Track> _poorScore;

protected Map<List<TrackerHit>, List<Track>> hitsToTracksMap;
protected Map<Track, List<Track>> sharedTracksMap;
protected Map<Track, double[]> trackScoreMap;

protected AmbiguityResolverUtils utils = new AmbiguityResolverUtils();
```

Internal state lists

- public get methods for internal state lists
- public void resetResolver()
- public void resolve()
- public double scoreTrack(Track track)
- protected int[] holesOnTrack(Track trk)
- protected boolean areShared(Track trk1, Track trk2)

hps-tracking ▶
src/main/java ▶
org.hps.recon.tracking ▶
AmbiguityResolver.java

Various ambiguity resolvers can inherit from this class

Can be called by MergeTrackCollections

SIMPLE AMBIGUITY RESOLVER

hps-tracking ▶ src/main/java ▶ org.hps.recon.tracking ▶ SimpleAmbiguityResolver.java

- Can be configured to replicate behaviour of old MergeTrackCollections
- Has a few more features as well
- Modes:
  - Remove duplicates
  - Remove partials
  - For tracks with too many shared hits (adjustable threshold), keep only the best-scoring track
  - Remove any tracks with poor score (adjustable threshold)
- Score = χ^2/dof

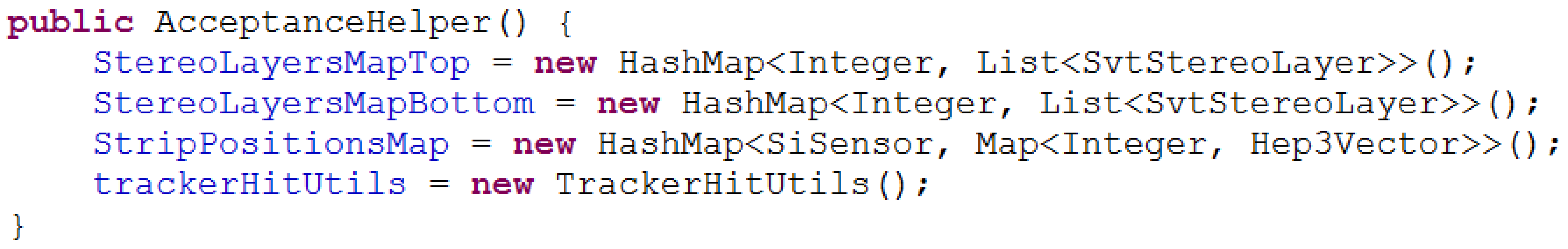
CLASSIC AMBIGUITY RESOLVER

hps-tracking ▶ src/main/java ▶ org.hps.recon.tracking ▶ ClassicAmbiguityResolver.java

- Simplified version of ATLAS Ambiguity Processor
- Remove duplicates, then score all tracks
- Loop through tracks, starting with highest-scoring tracks. For each track, remove all lower-scoring tracks that share too many hits (adjustable threshold) with it
- Scoring system:
  - (adjustable by layer) points for each unshared hit
  - (adjustable by layer) points for each shared hit
  - (adjustable by layer) penalty for each hole
  - (adjustable by layer) penalty for going outside layer acceptance
  - define cumProb = ChisqProb.*gammp(track.getNDF(), track.getChi2())*
    - if better than (adjustable) threshold, add *(adjustable factor) x* | log_{10}*(cumProb)* | to score
    - otherwise, subtract (adjustable) penalty from score

CLASSIC AMBIGUITY RESOLVER

- Scoring system requires determination of holes and missed-acceptances, taking into account bad channels
- Facilitated by new AcceptanceHelper class in hps-tracking ► src/main/java ► org.hps.recon.tracking



```
public AcceptanceHelper() {
    StereoLayersMapTop = new HashMap<Integer, List<SvtStereoLayer>>();
    StereoLayersMapBottom = new HashMap<Integer, List<SvtStereoLayer>>();
    StripPositionsMap = new HashMap<SiSensor, Map<Integer, Hep3Vector>>();
    trackerHitUtils = new TrackerHitUtils();
}
```

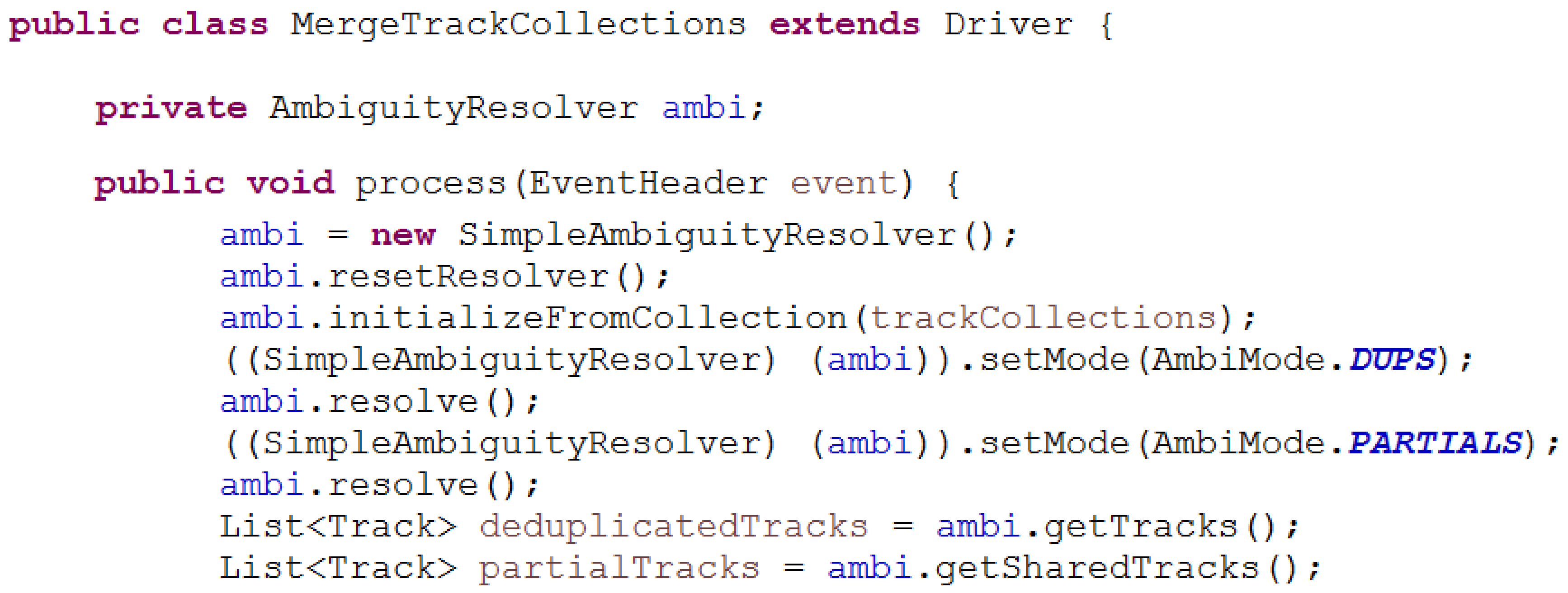
- `protected boolean isWithinAcceptance(Track trk, int layer)`
- `public int findIntersectingChannel(Hep3Vector trackPosition, SiSensor sensor)`

POSSIBILITIES FOR MORE AMBIGUITY RESOLVERS

- Classic + Track-cleaning
  - Requires outlier detection
  - May include hit recovery (for holes)
- Classic + Cluster-sharing probability
  - Takes into account probability that a given cluster is shared, based on cluster properties
- Classic + Hit timing
  - Includes hit timing in χ^2 in scoring
- Simulated Annealing
  - Designed for "left/right" ambiguity resolution between tracks that share many hits
- Elastic Neural Net
  - Probably overkill

TESTING SIMPLE AMBIGUITY RESOLVER

- Made new version of MergeTrackCollections with SimpleAmbiguityResolver



```
public class MergeTrackCollections extends Driver {

    private AmbiguityResolver ambi;

    public void process(EventHeader event) {
          ambi = new SimpleAmbiguityResolver();
          ambi.resetResolver();
          ambi.initializeFromCollection(trackCollections);
          ((SimpleAmbiguityResolver) (ambi)).setMode(AmbiMode.DUPS);
          ambi.resolve();
          ((SimpleAmbiguityResolver) (ambi)).setMode(AmbiMode.PARTIALS);
          ambi.resolve();
          List<Track> deduplicatedTracks = ambi.getTracks();
          List<Track> partialTracks = ambi.getSharedTracks();
```

- Created hps-tracking ▶ src/test/java ▶ org.hps.recon.tracking ▶ MergeTrackCollectionsTest to perform raw → reconstructed lcio with old vs new MergeTrackCollections
- Examined reco lcio's using DQM : plots look the same ☺

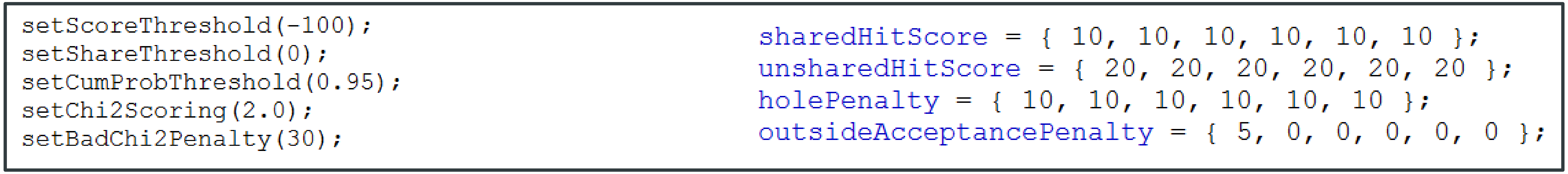
TESTING CLASSIC AMBIGUITY RESOLVER: ACCEPTANCE HELPER

- hps-users ▶ src/main/java ▶ org.hps.users.mdiamond ▶ HoleCreationDriver
  - From each track in reco lcio file:

  1. Removes hits (makes holes) on track according to pre-set pattern, or at random
  2. Makes new TrackerHitCollection, RawTrackerHitCollection, HelicalTrackHitCollection, and RotatedHelicalTrackHitCollection that exclude those hits

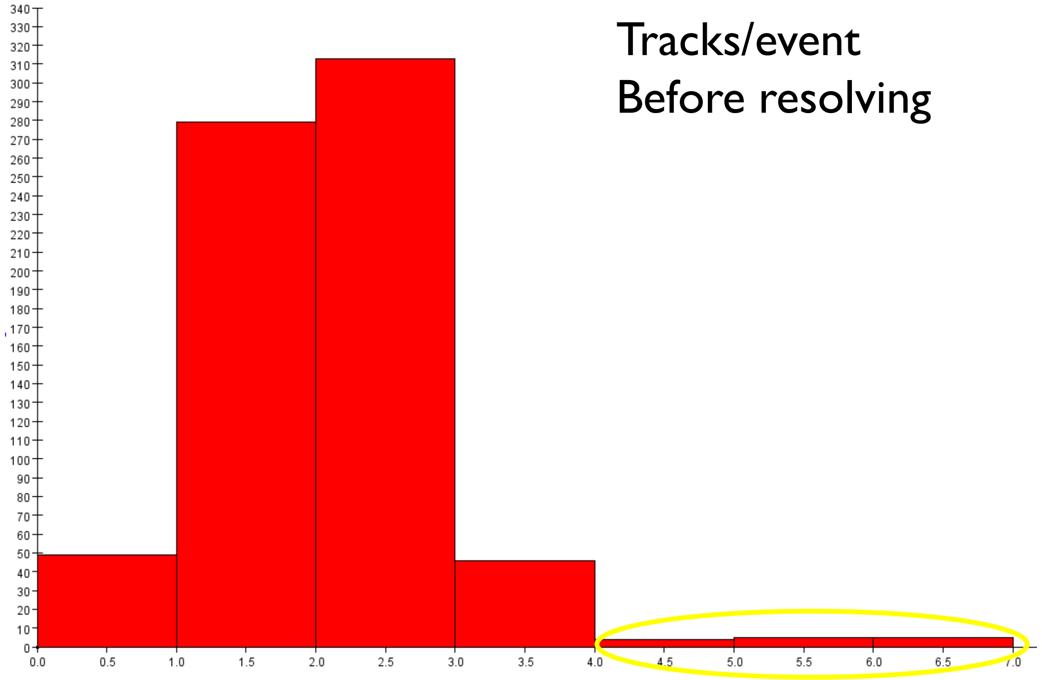
  - Outputs to new reco lcio file
- ClassicAmbiguityResolver's holesOnTrack, using AcceptanceHelper, correctly identified holes in tracks in new lcio ☺

TESTING CLASSIC AMBIGUITY RESOLVER: MC SAMPLE OF 700 A’ EVENTS

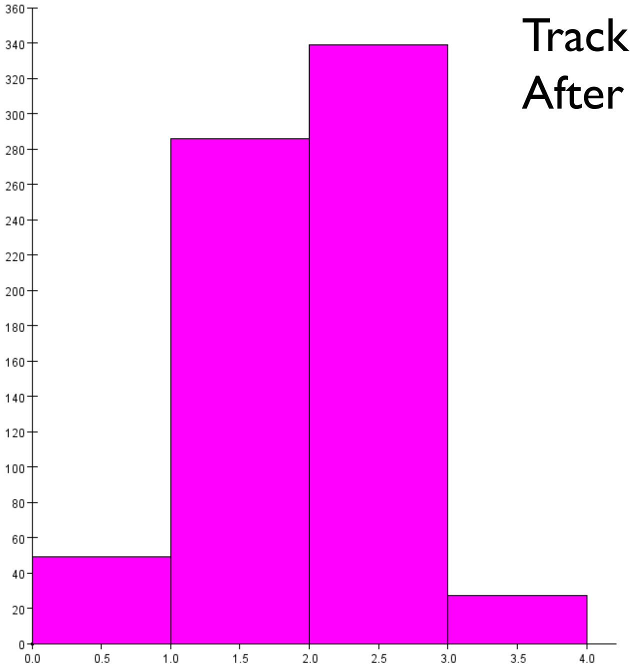


```
setScoreThreshold(-100);
setShareThreshold(0);
setCumProbThreshold(0.95);
setChi2Scoring(2.0);
setBadChi2Penalty(30);
```

```
sharedHitScore = { 10, 10, 10, 10, 10, 10 };
unsharedHitScore = { 20, 20, 20, 20, 20, 20 };
holePenalty = { 10, 10, 10, 10, 10, 10 };
outsideAcceptancePenalty = { 5, 0, 0, 0, 0, 0 };
```

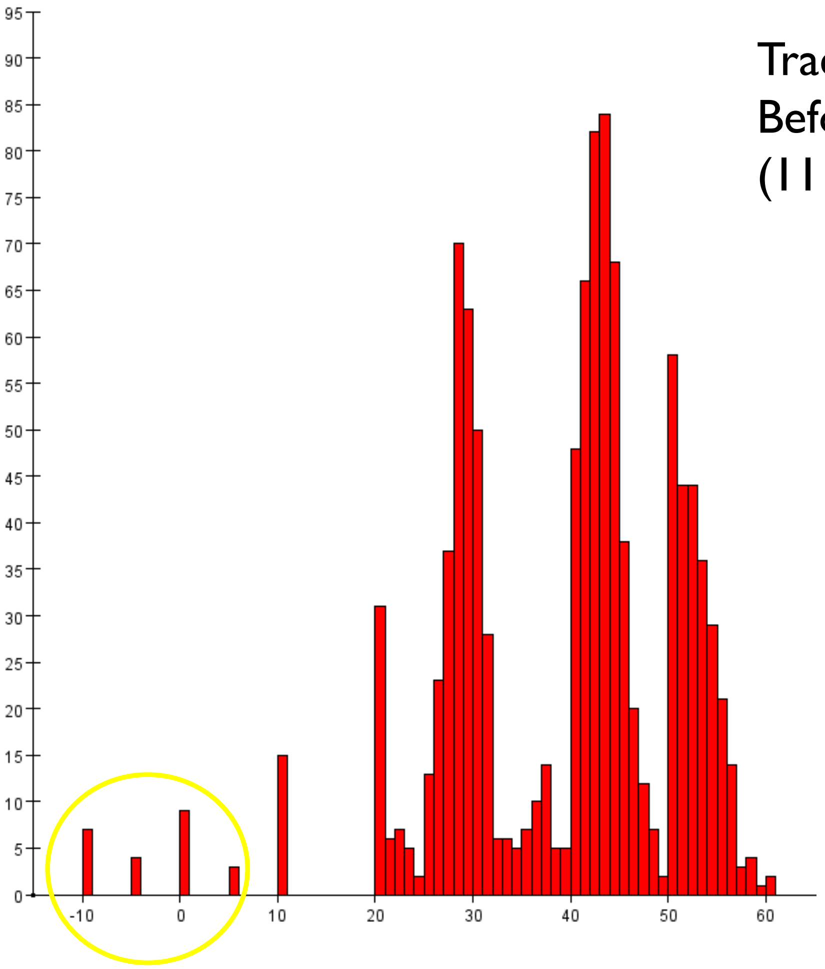


Tracks/event
Before resolving

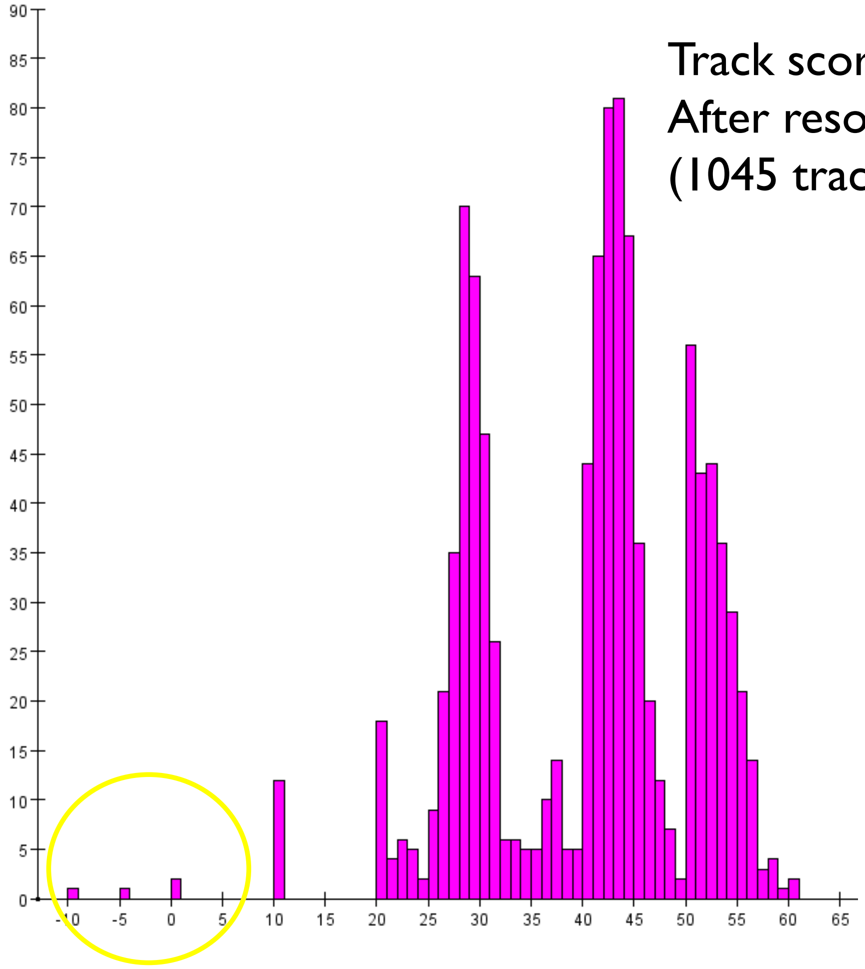


Tracks/event
After resolving

TESTING CLASSIC AMBIGUITY RESOLVER: MC SAMPLE OF 700 A’ EVENTS



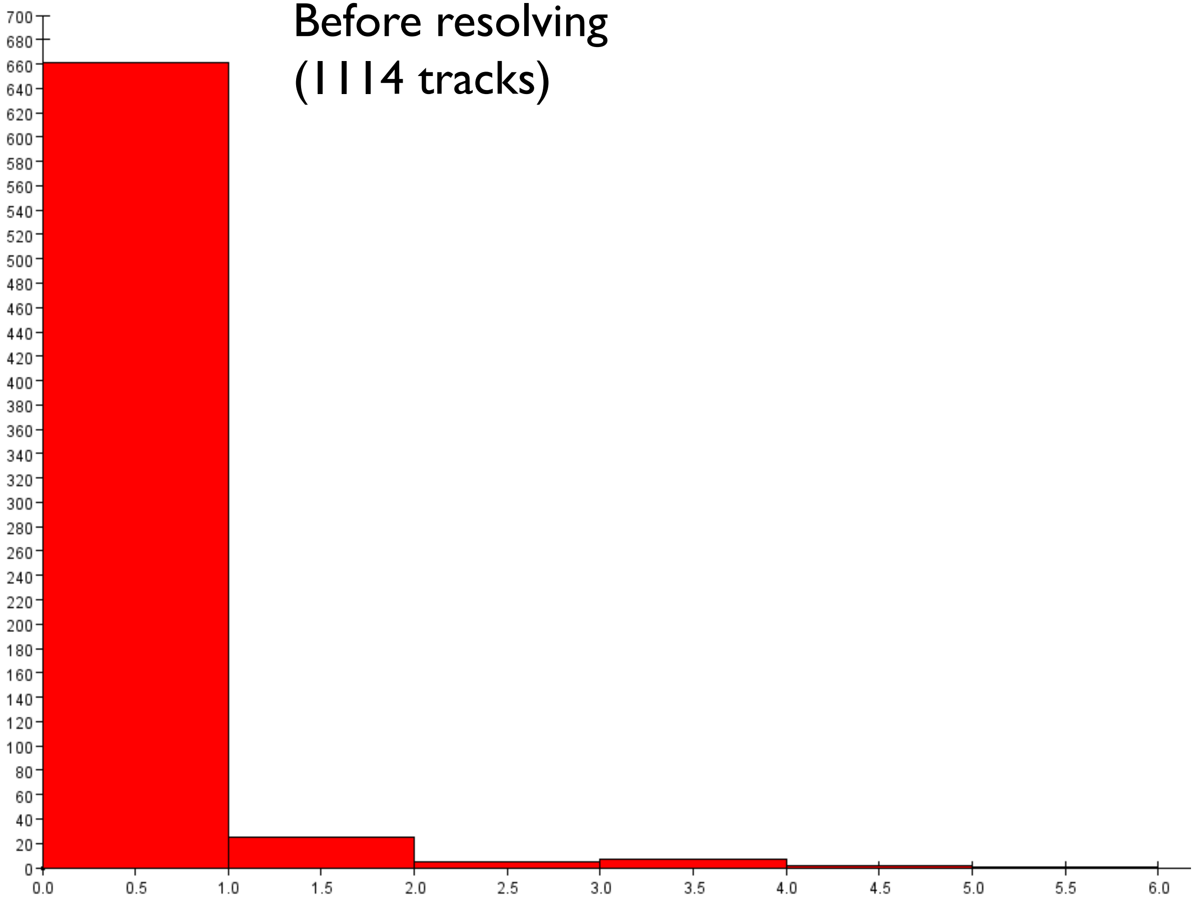
Track scores
Before resolving
(1114 tracks)



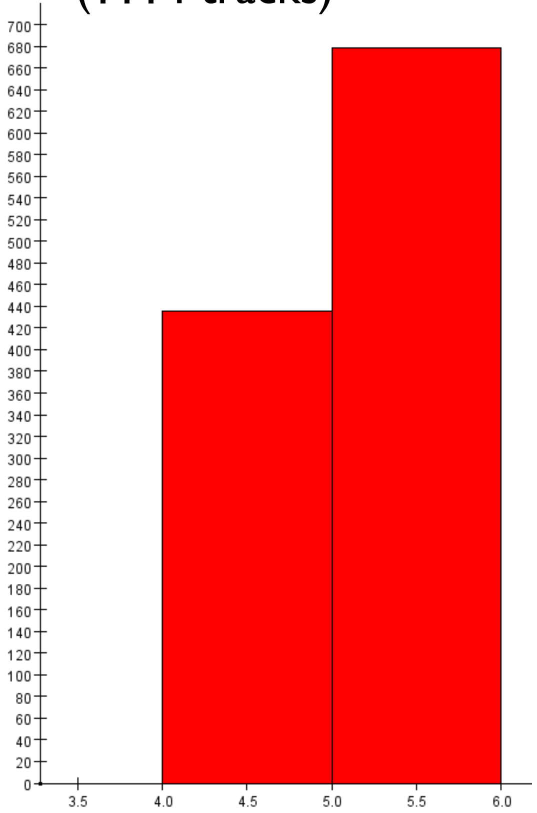
Track scores
After resolving
(1045 tracks)

TESTING CLASSIC AMBIGUITY RESOLVER: MC SAMPLE OF 700 A’ EVENTS

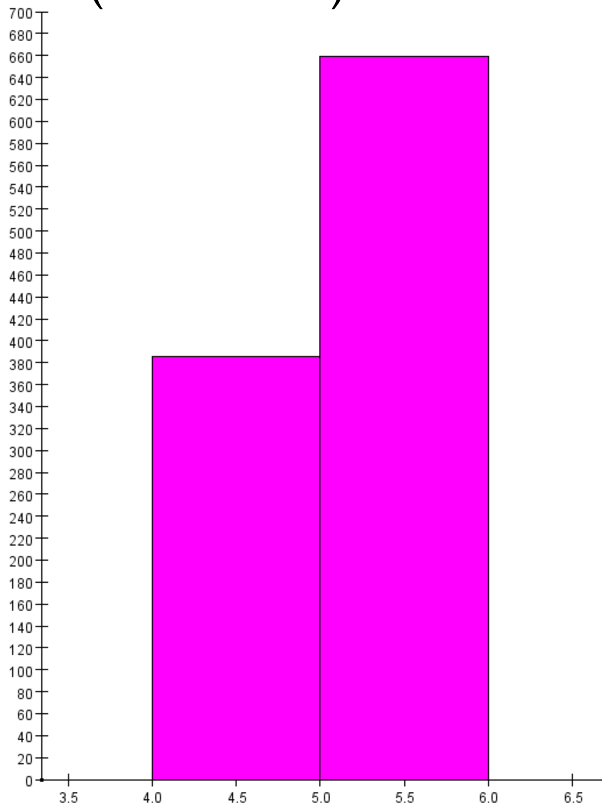
Shared hits/track
Before resolving
(1114 tracks)



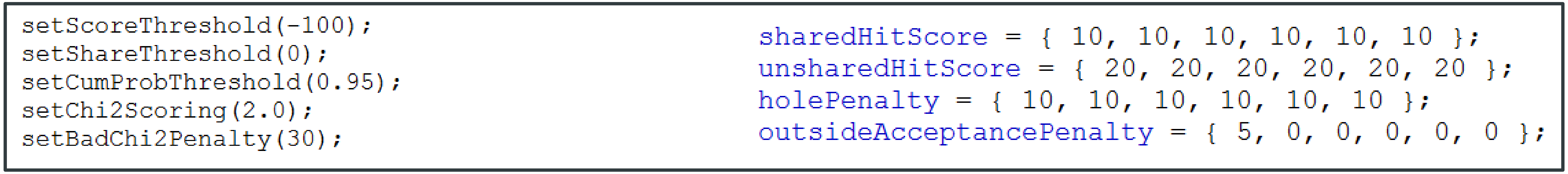
Hits/track
Before resolving
(1114 tracks)



Hits/track
After resolving
(1045 tracks)

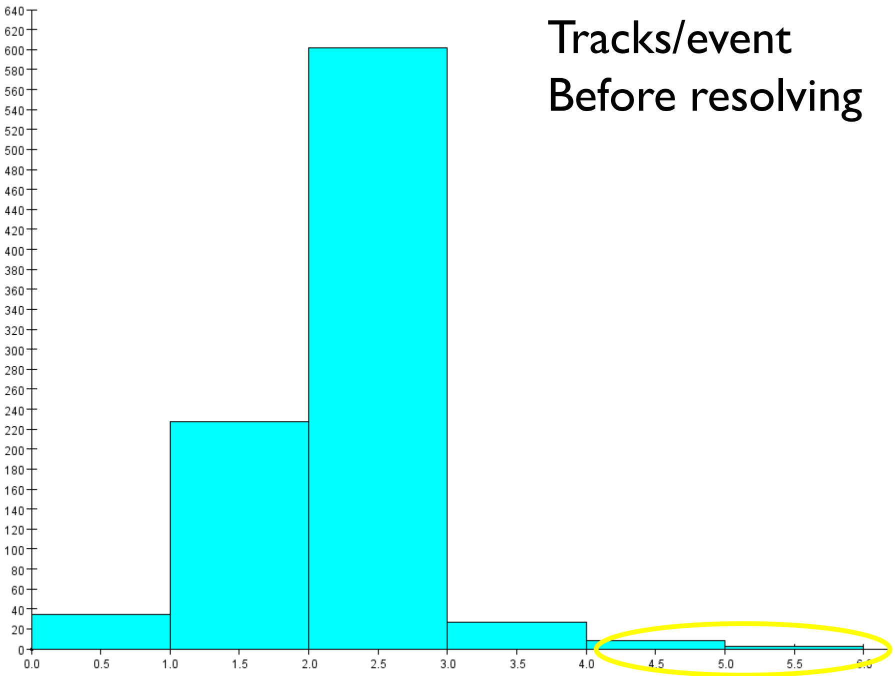


TESTING CLASSIC AMBIGUITY RESOLVER: MC SAMPLE OF 900 TRITRIG EVENTS

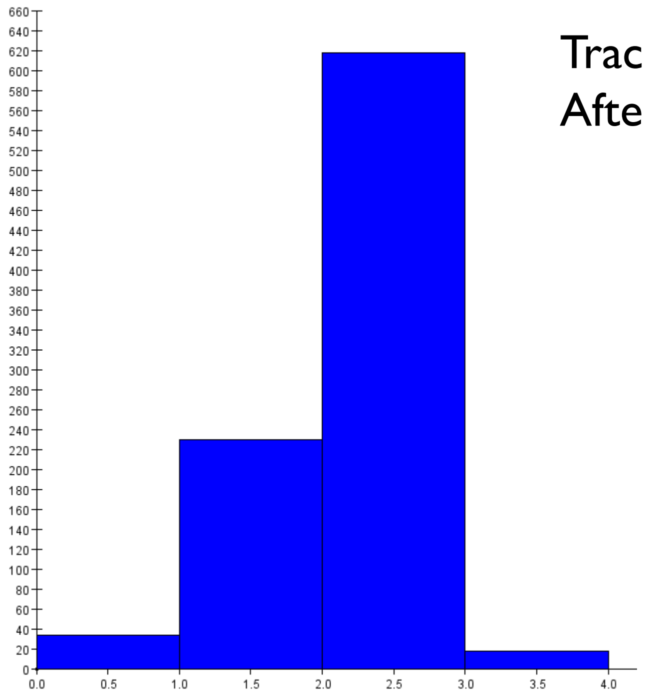


```
setScoreThreshold(-100);
setShareThreshold(0);
setCumProbThreshold(0.95);
setChi2Scoring(2.0);
setBadChi2Penalty(30);
```

```
sharedHitScore = { 10, 10, 10, 10, 10, 10 };
unsharedHitScore = { 20, 20, 20, 20, 20, 20 };
holePenalty = { 10, 10, 10, 10, 10, 10 };
outsideAcceptancePenalty = { 5, 0, 0, 0, 0, 0 };
```

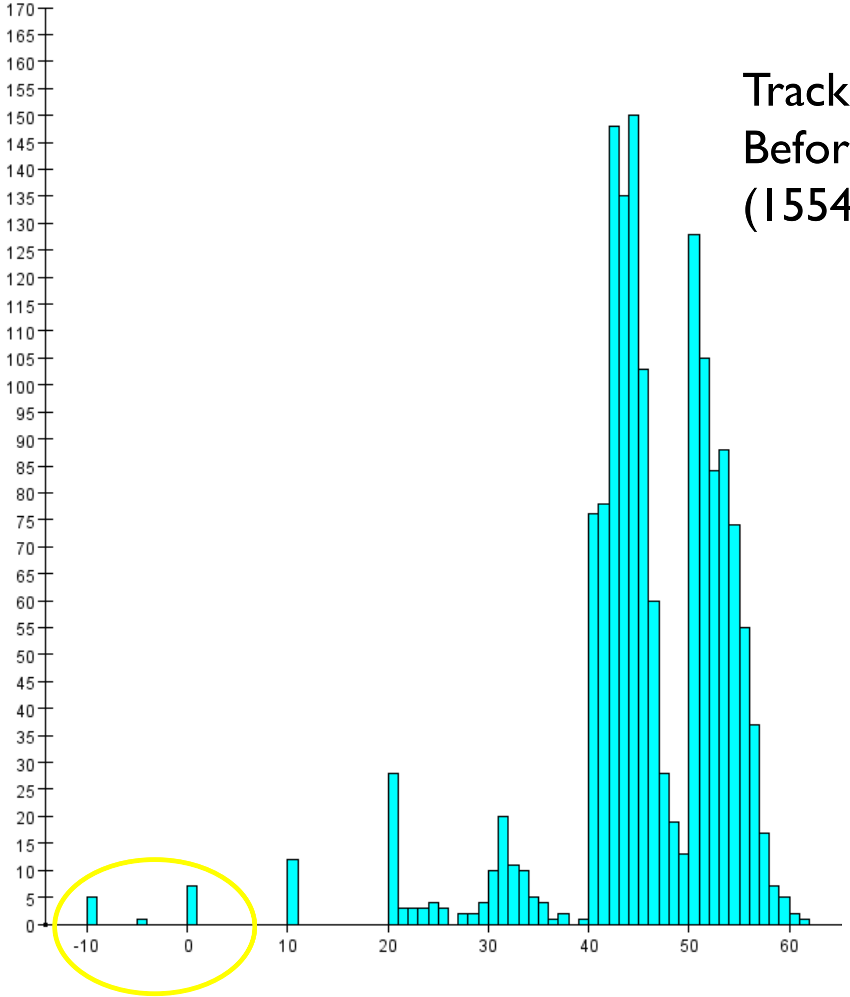


Tracks/event
Before resolving

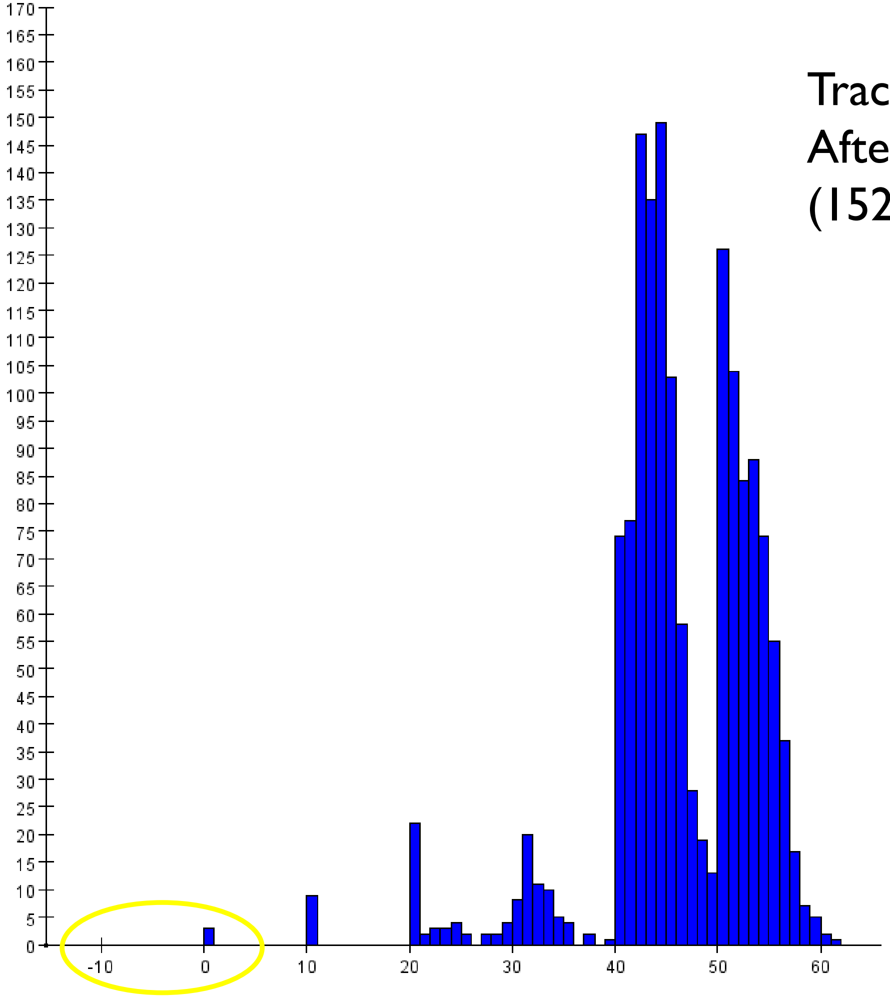


Tracks/event
After resolving

TESTING CLASSIC AMBIGUITY RESOLVER: MC SAMPLE OF 900 TRITRIG EVENTS

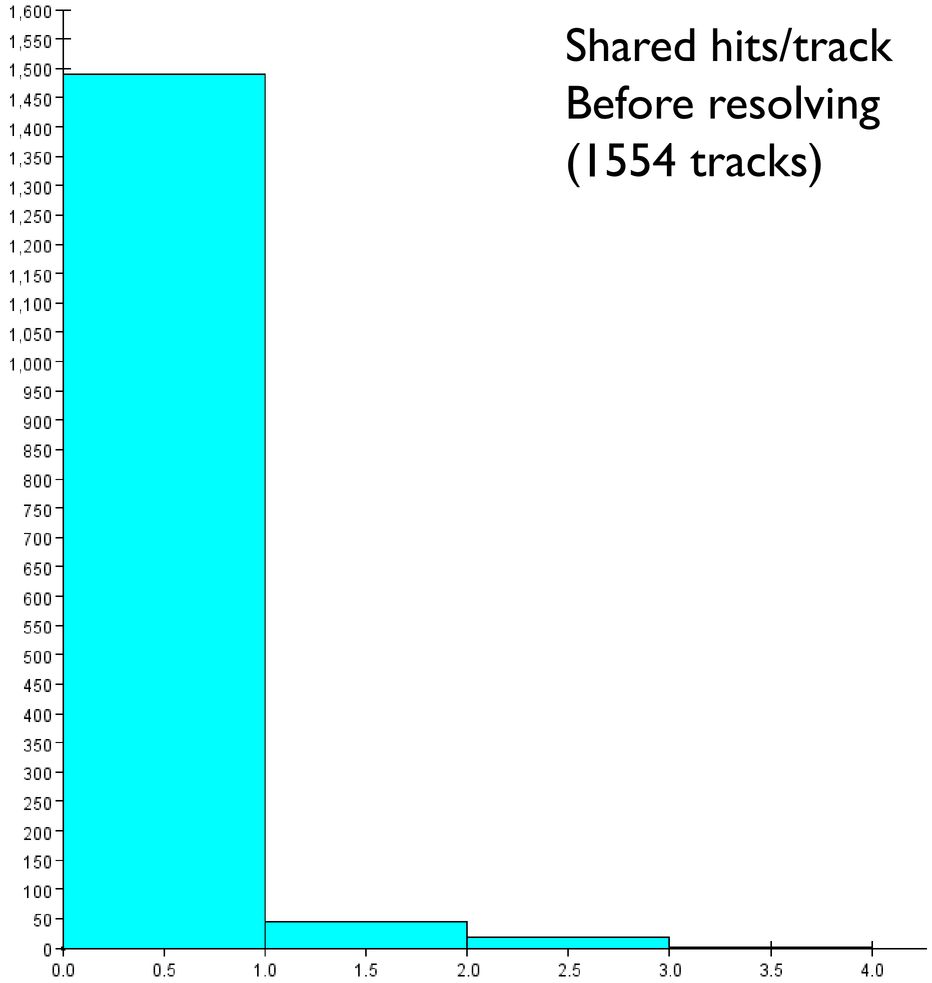


Track scores
Before resolving
(1554 tracks)



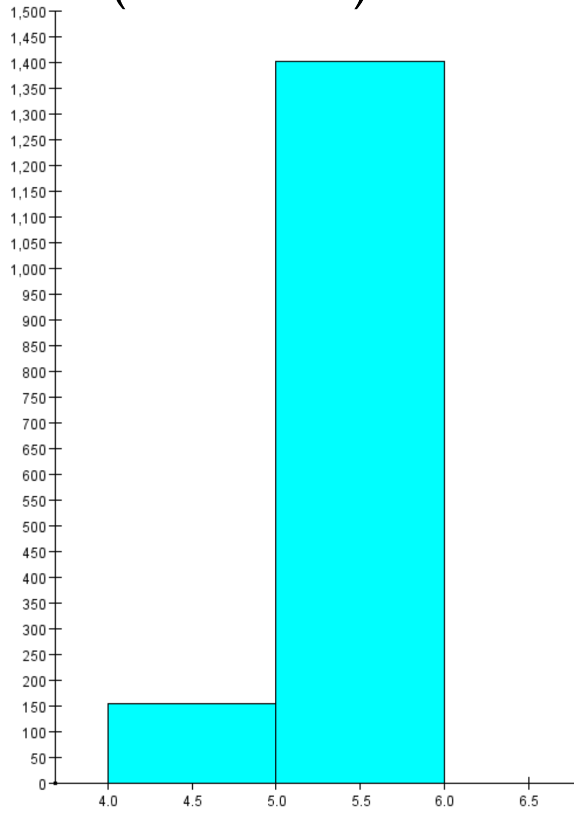
Track scores
After resolving
(1520 tracks)

TESTING CLASSIC AMBIGUITY RESOLVER: MC SAMPLE OF 900 TRITRIG EVENTS

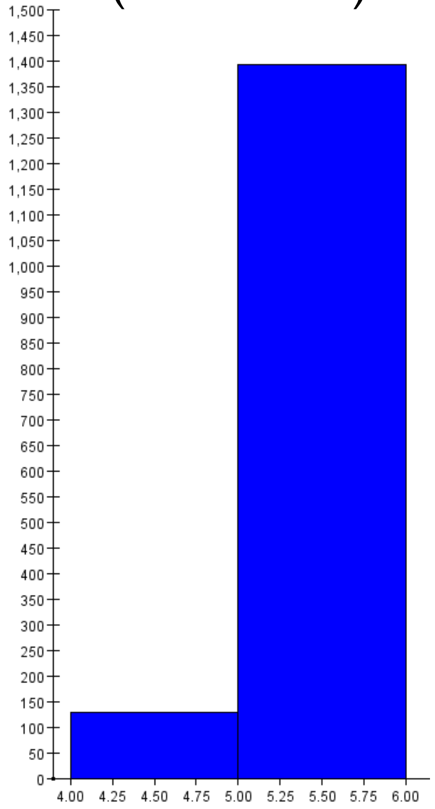


Shared hits/track
Before resolving
(1554 tracks)

Hits/track
Before resolving
(1114 tracks)



Hits/track
After resolving
(1520 tracks)



IDEAS FOR STUDIES & TESTING ?

- Events that share all but one or two hits
  - Generate from scratch, using particle gun with two very-nearby particles
- Artificially high-multiplicity events
  - How to generate?
- Optimizing parameters in track-scoring
  - How? Data-driven or MC-driven?
- High-statistics runs
  - On what?

TRACK CANDIDATE IDENTIFICATION: PRIORITIES FOR IMPROVEMENT

JUNE 5 2016

I THINK WE HAVE TWO OPTIONS …

1. Improve the existing framework
   - optimize sectoring scheme
   - optimize triplet-finding in SeedTrackFinder/FastCheck, using common ATLAS/CMS algorithms
   - improve track seed extension/confirmation/re-fitting in ConfirmerExtender, probably with Kalman filtering techniques
     - fix/eliminate slow helix-plane intersection finding, identified by Maurik

2. Replace existing framework with cellular automaton + track-following system
   - optimize pair-finding
   - use hit pairs as units of the cellular automaton
   - implement track-following to "jump over holes" (which cellular automaton can't do by itself)