StvKNN

V. Perevoztchikov
Brookhaven National Laboratory,USA

K-Nearest Neighbor(KNN) seed finder

K Nearest Neighbor is one of those algorithms that are very simple to understand but works incredibly well in practice (Saravan Thirumuruganathan)

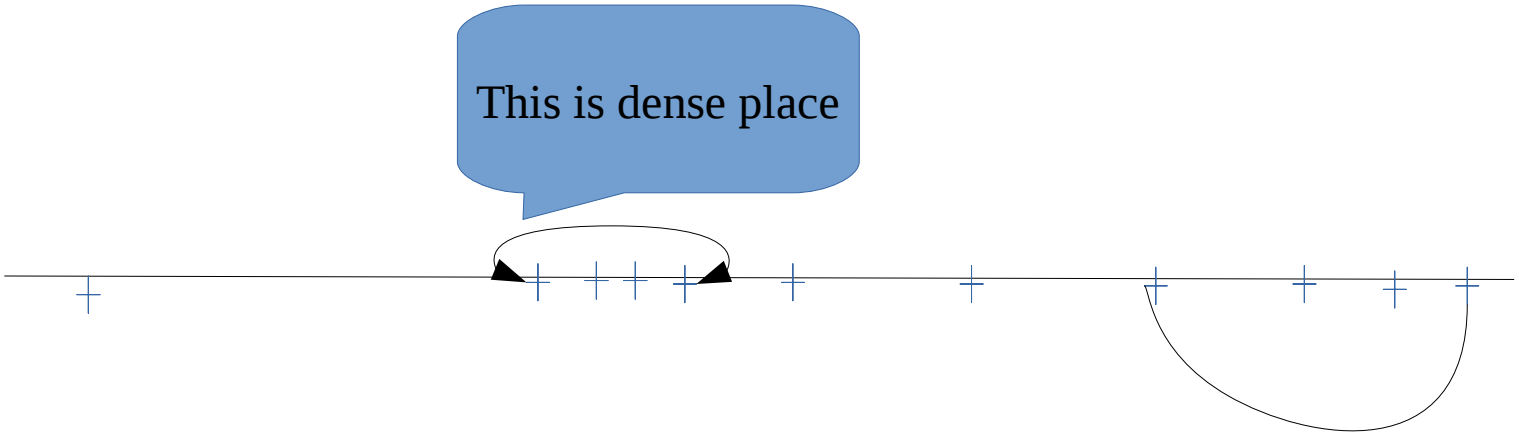
http://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm

The site above gives rather good description of KNN method.

Short description of KNN seed finder. The main idea, if you look along the straight track, you see the dot. In reality track is not a straight, but for small part of it, it is close to that.

1. Select first hit;
2. Direction from hit to the bisection of vertex Z range is the X axis in spherical coordinate system;
3. Arbitrary define orthogonal axis's Y and Z
4. Select not too big volume near first hit;
5. Loop over hits in volume , calculate Phi and Theta;
6. Calculate distances in (Phi,Theta) between the hits;
7. Save the Kth smallest distances for each hit;
8. Select hit with the smallest K distance;
9. This hit and the nearest ones forms the seed.

Simple one dimension KNN example



Dense place here is a place where distance to closest 4th point is minimal

KNN seed finder(2)

If found KNN distance is too big, there is no seed.

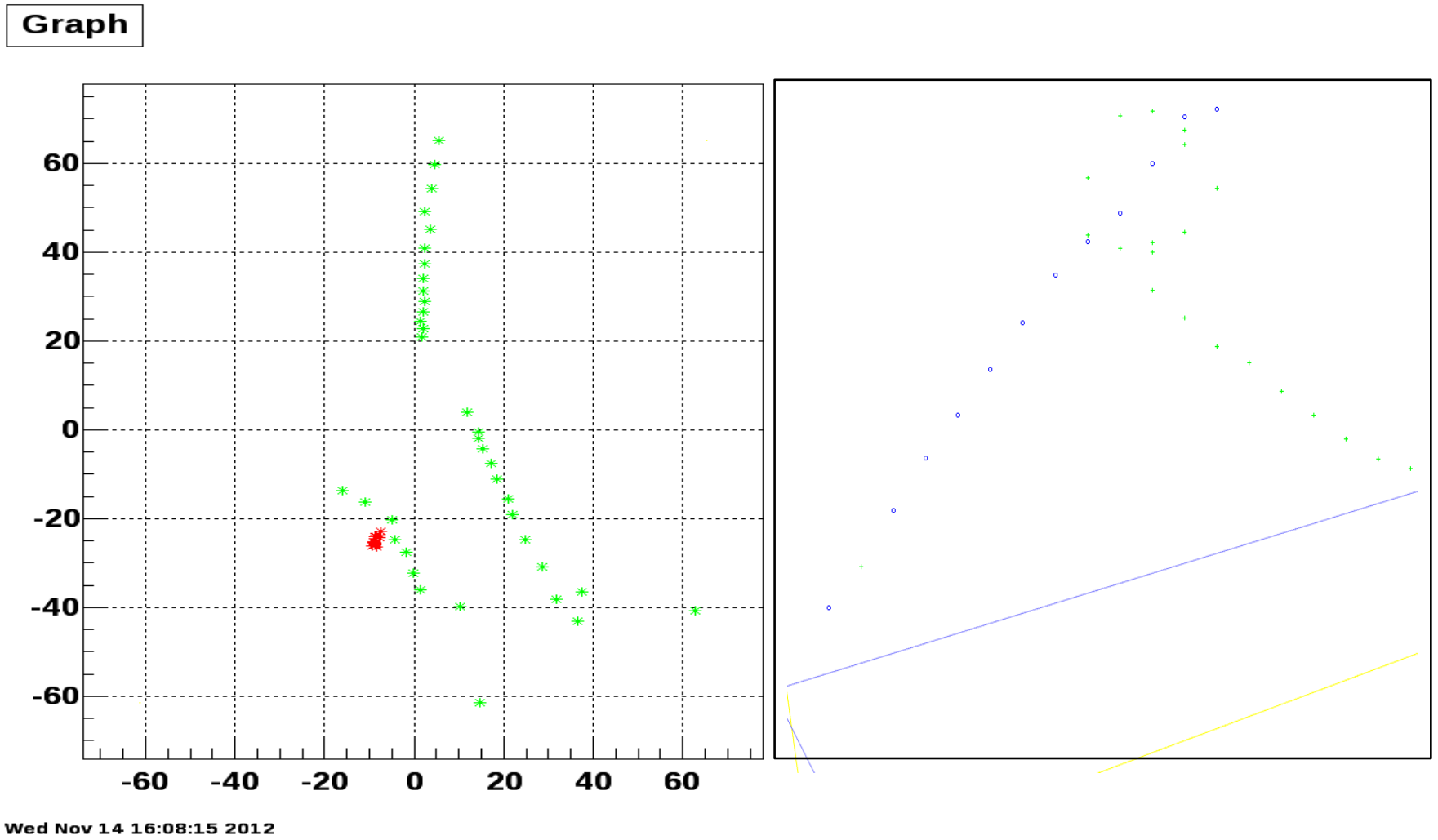
If track is not straight, then in (Phi,Theta) space it will be not a dot but short line.

Calculate the width of this line. If width is too big, it is not a seed.

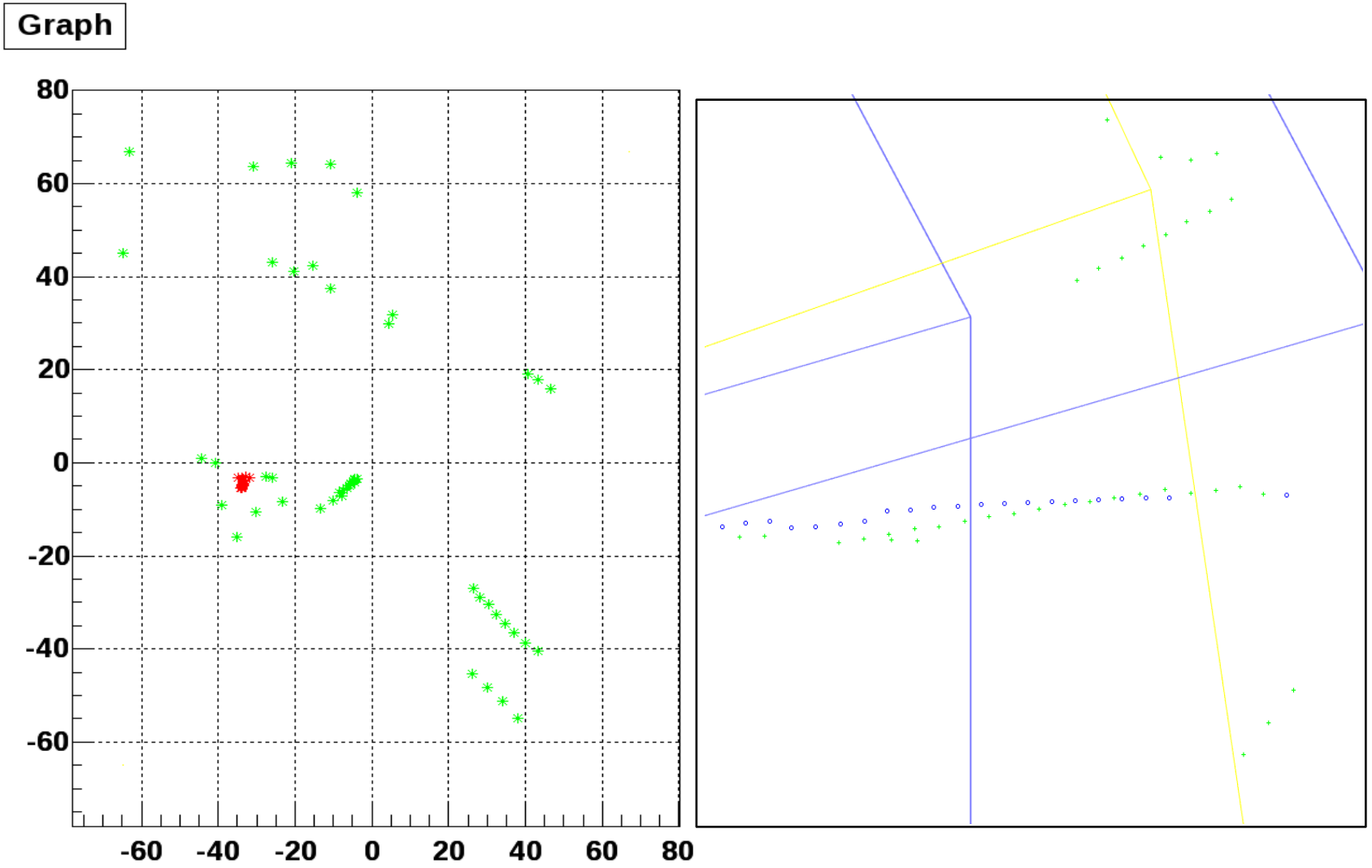
On the next page two figures of good seed.

Afterwards there is an example of bad seed.

Good KNN seed

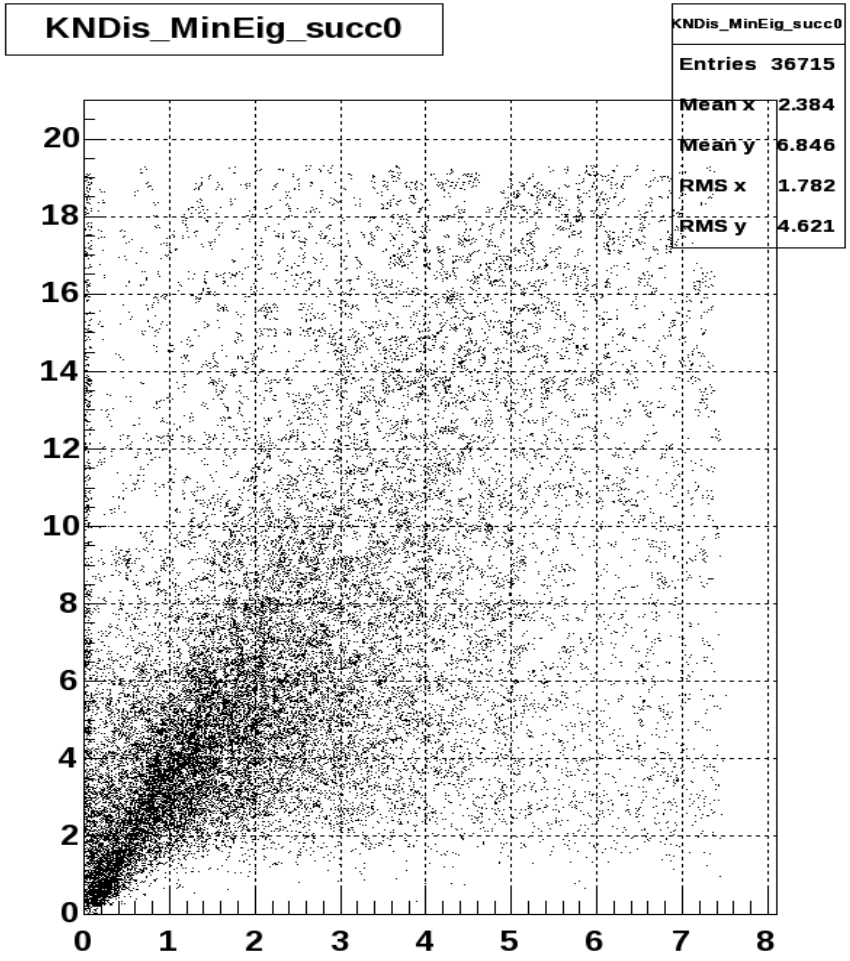


KNN bad seed

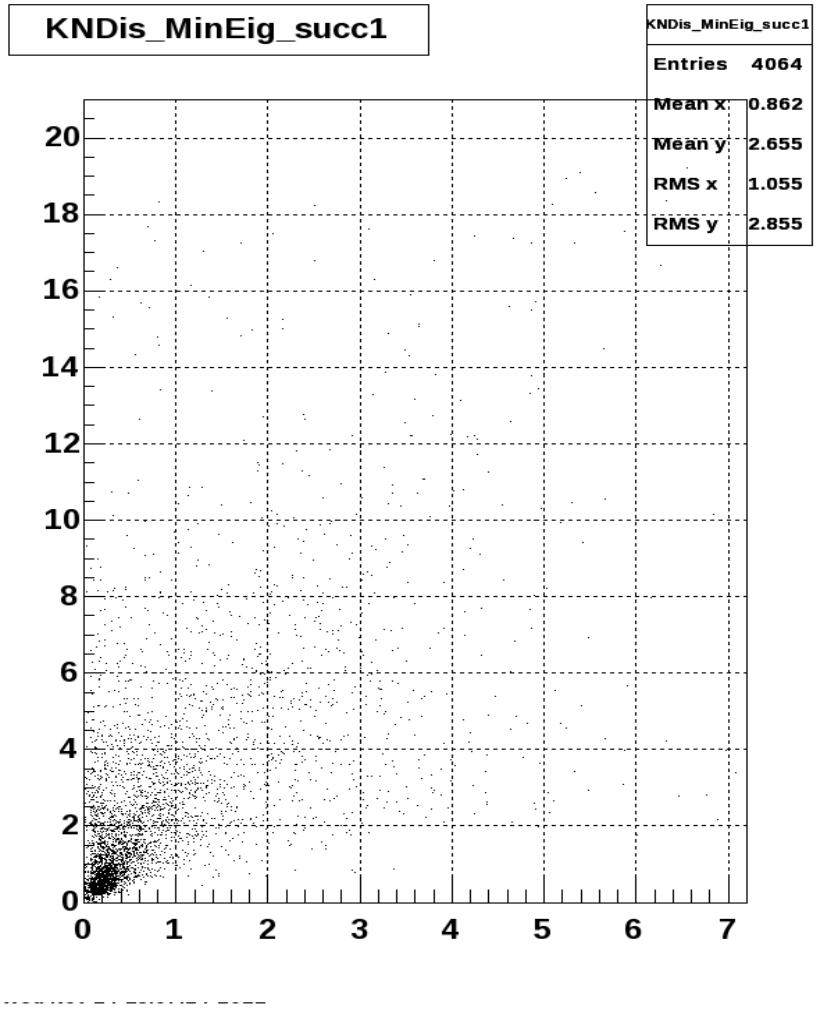

Graph
80
60
40
20
0
-20
-40
-60
-60
-40
-20
0
20
40
60
80
Wed Nov 14 16:15:31 2012

KNN statistics wide cuts

No tracks

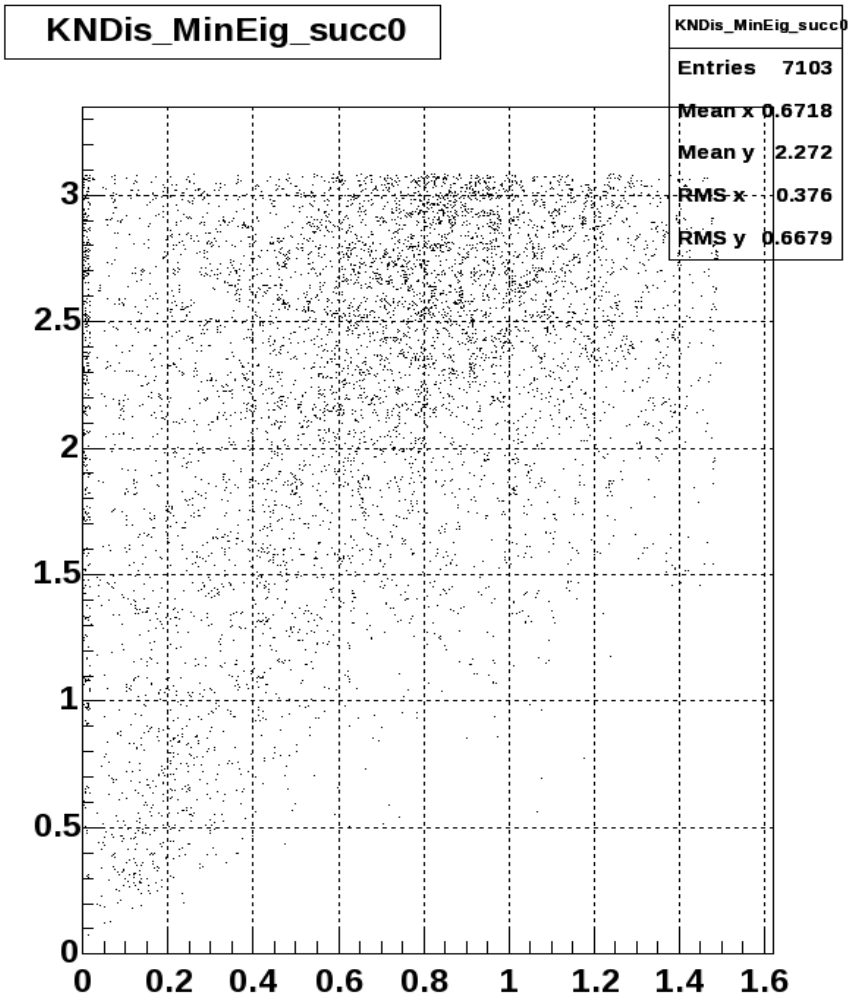


Found tracks

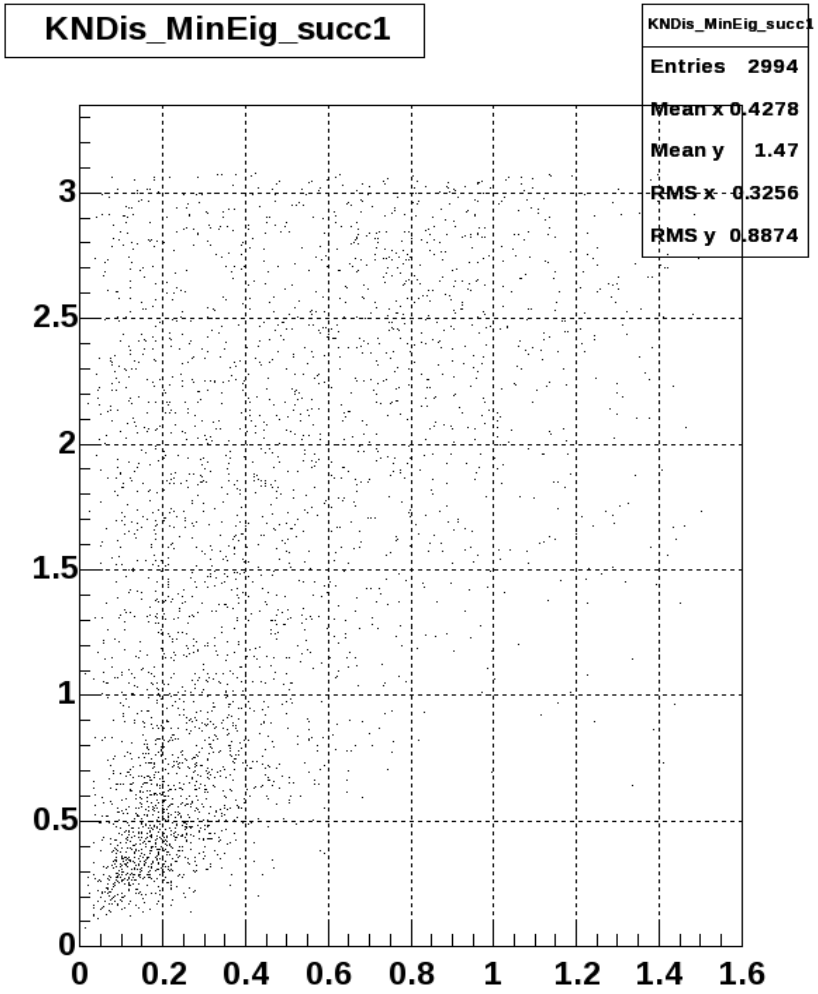


KNN statistics narrow cuts

No tracks



Found tracks

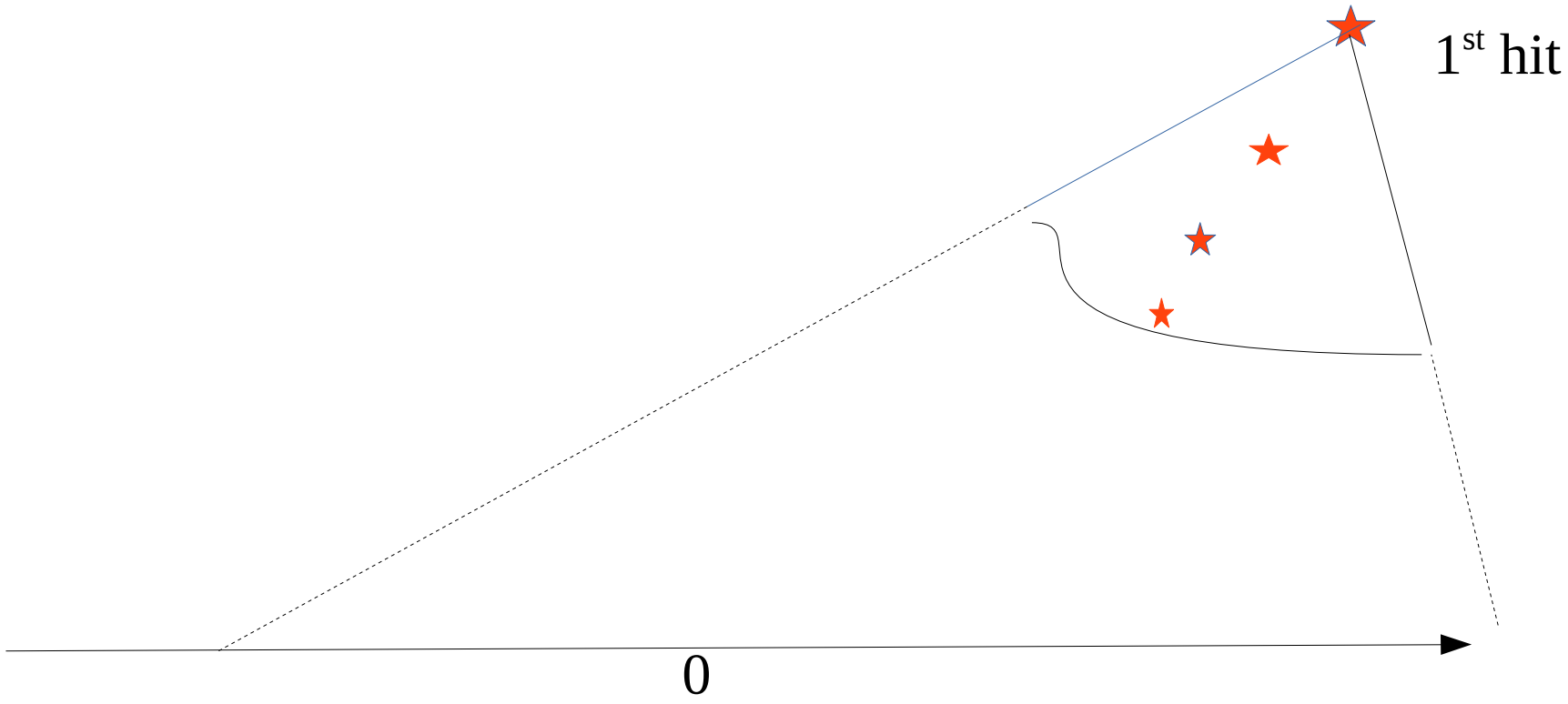


Detailed algorithm

Full algorithm consists of parts:

- Selection of first hit;
- Defining the spherical cone containing all the needed hits;
- Defining the box surrounding the cone;
- Selection of the hits inside the box;
- Selection of the hits inside the cone;
- Calculation of all the distances between the hits, and finding one where the distance to the closest Kth hit is the smallest;
- Selection set hits closest to found one and use this set as a seed;

Cone



Step by step

- Selection of first hit:
  - Once per event created hit container ordered by distance to zero. It is first hits container. First hit selected in loop, skipping used hits. Then iterator is saved for next event;
- Defining the spherical cone containing all the needed hits:
  - Top point of cone in the first hit.
  - Direction towards to Z axis
  - Opening angle selected to include Z= +-220
  - Height(radius) to include 5-10 detector planes.
- Defining the box surrounding the cone:
  - The minimal box to include all the points of cone.
- Selection of the hits inside the box;
- Selection of the hits inside the cone;
- Calculation of the all distances between the hits and finding one where the distance to closest Kth hit is the smallest;
- Selection set hits closest to found one and use this set as a seed;
- Fit by helix.

Step by step(continue.1)

- Selection of the hits inside the box:
  - Once per event created container with 3 (x,y,z) or 5(x,y,z,x-y,x+y) keys. This container allows to search hits limited by defined box in these coordinates. Objects in this container is somehow ordered and search is rather fast. Using box, surrounding the cone, we got the set of hits.
- Selection of the hits inside the cone:
  - Select hits which inside the cone.

Step by step(continue.2)

- Calculation of the all distances between the hits and finding one where the distance to closest Kth hit is the smallest:
  - It is simple to do by combinatorics, but it is slow,proportional to N**2;
  - It is clear if two hits are very far from each other, no sense to calculate distance. To do this possible for each hit was defined two angles, Theta and Phi. A std::map container sorted by int(Theta/step) was introduced, where each entry is a std::map container sorted by Phi. It allows to skip from comparing hits

    which are far in Theta or Phi. During comparison, the most dense hit is remembered. More precise, distance between hit and fourth closest hit is the smallest;
- Selection set hits closest to found one and use this set as a seed:
  - Starting from this hit, search of close hits recursively, adding to set.

Step by step(continue.3)

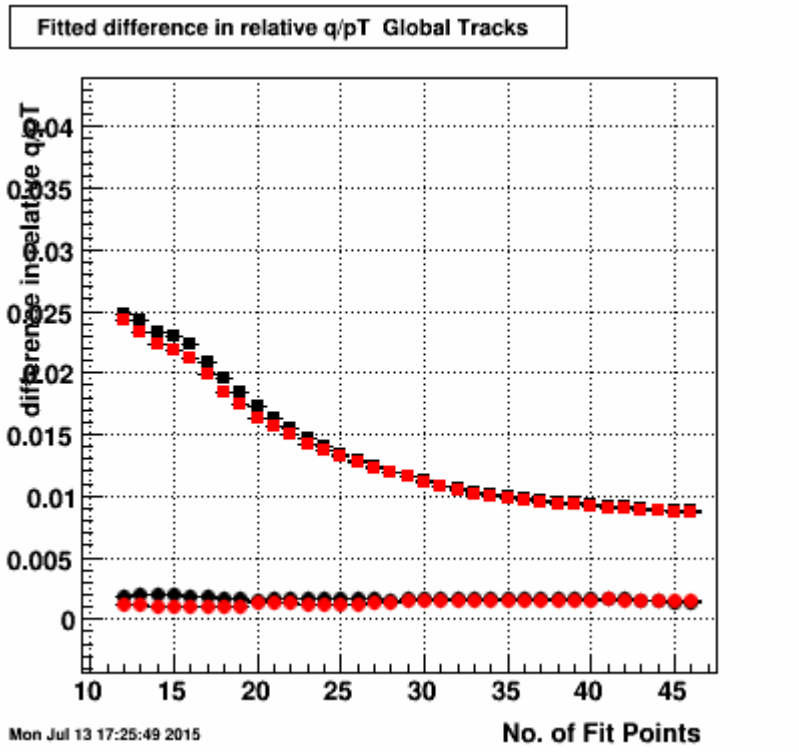
- Fit by helix:
  - All the hits sorted by distance to first hit;
  - When there are hits from the same detector, out-liners removed;
  - Fit the set by helix. When there are hits with very big Xi2, they are removed and refitted again
- Seed is ready.

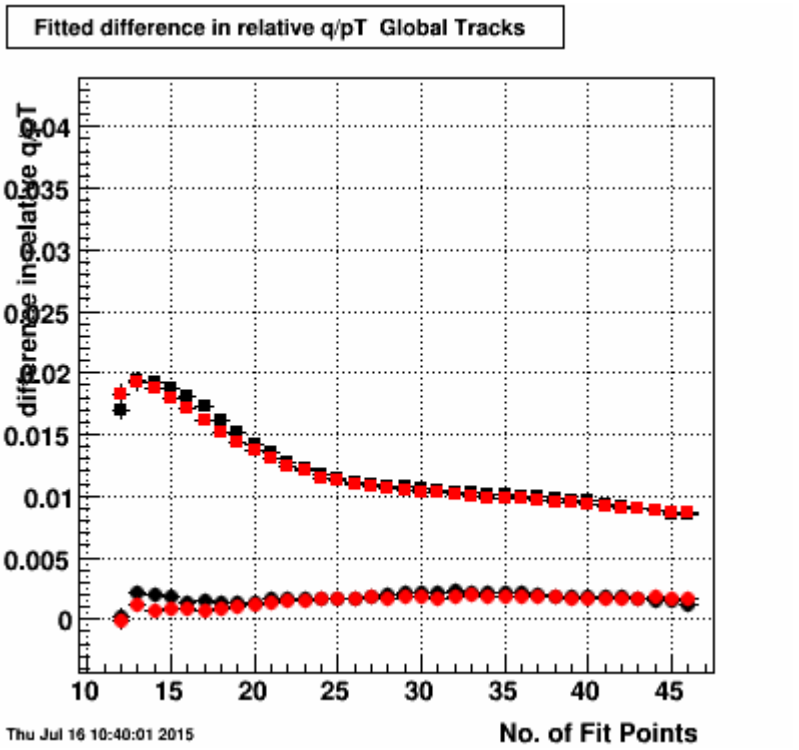
Summary

- Good features:
  - Ratio seeds/tracks = 7/10, in standard seed finder 1/10
  - Decision based on 5-10 hits, not on two as in standard seed-finder.
- Bad features:
  - Efficiency still less (-20%), not solved yet
  - Speed is still less (-10%)

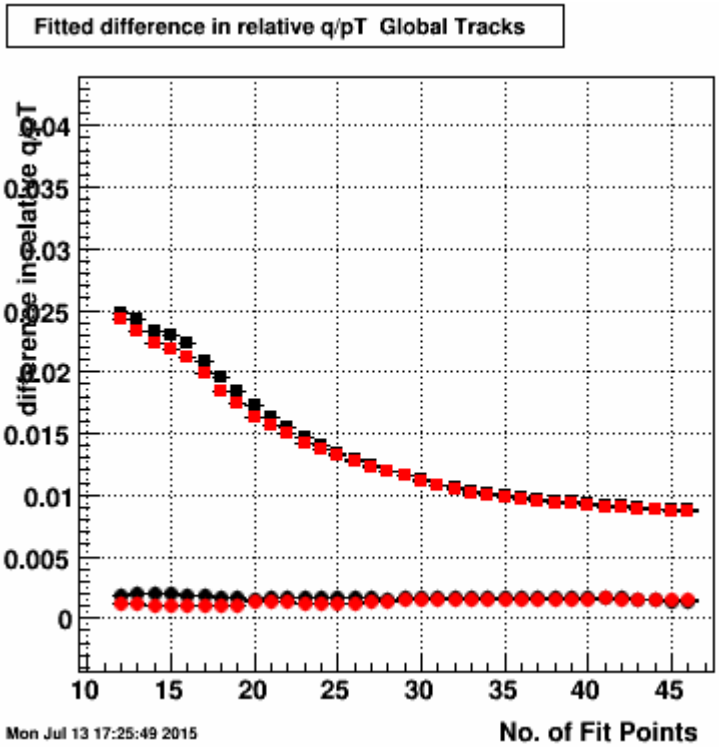
So evident that job is not done yet, but it is frozen. Working with HFT and FTS we must tune the standard seed finder first. In the same time, I am sure that for FTS only KNN could work properly. It is related to search the nearest hit in standard seed finder. But for FTS all the hits on the next layer almost “closest”. Hence to select the best almost impossible

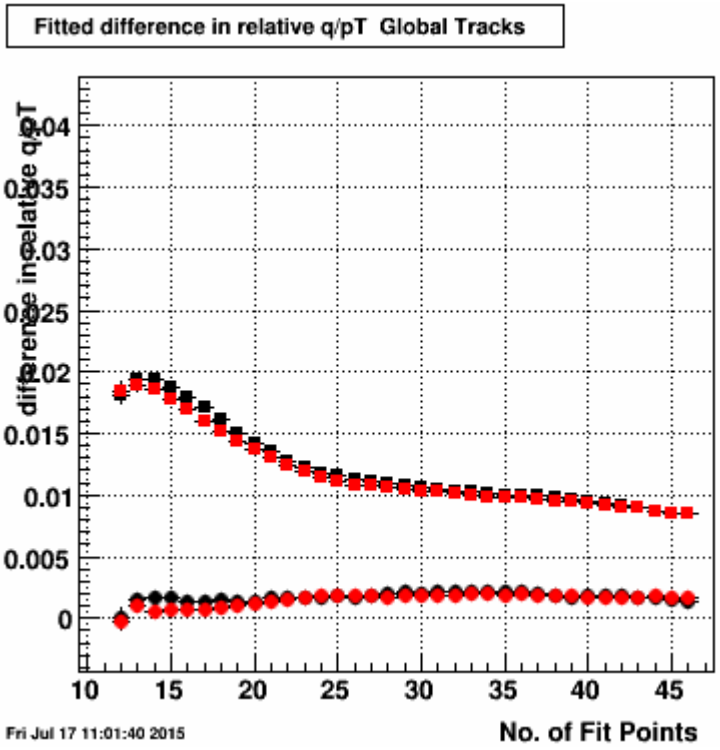
Example 1. Knn Better



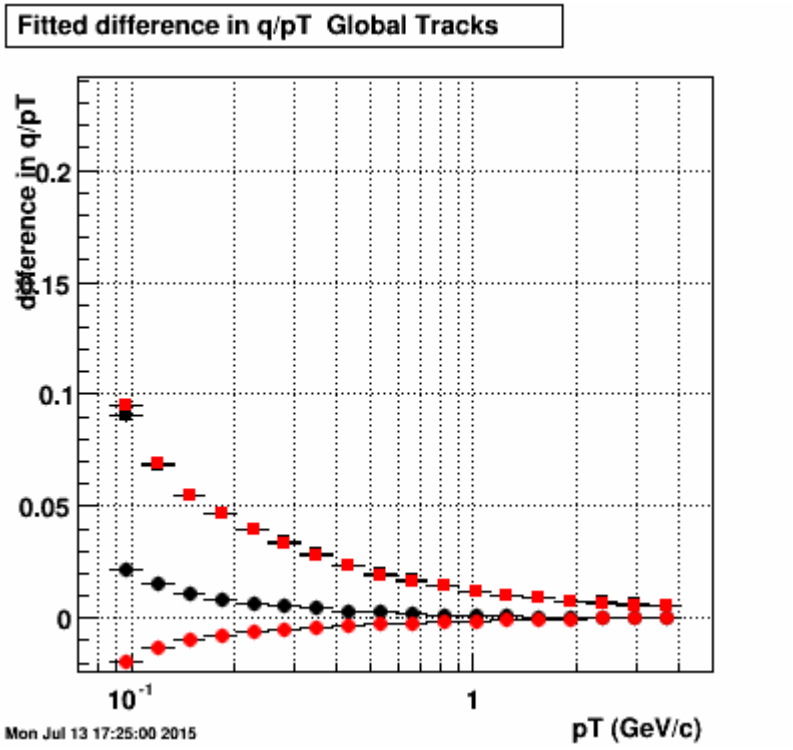


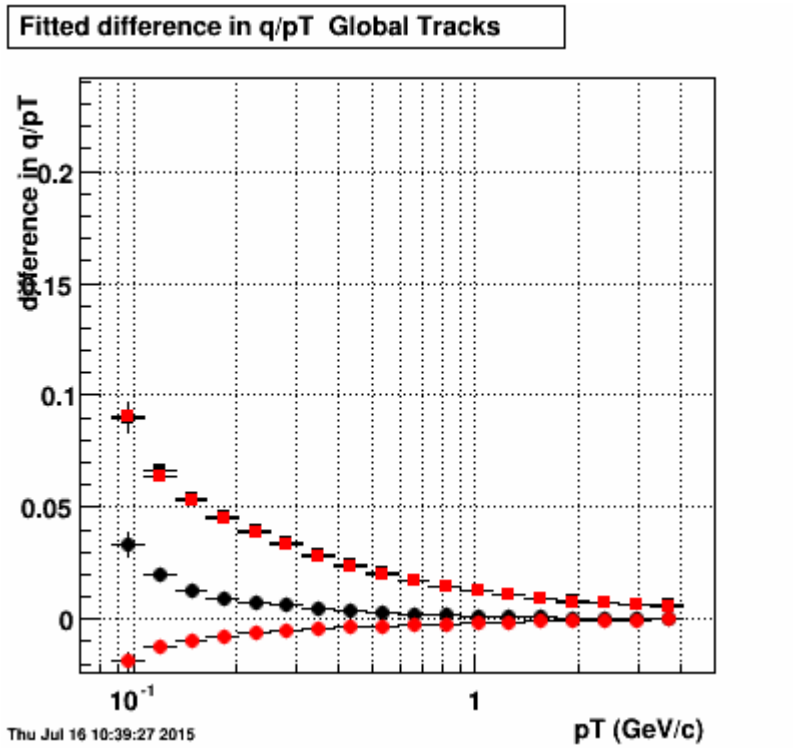
Example 2. Knn better





Example 3. Knn worse





Example 4. Knn WORSE