The BRAHMS Cluster algorithm.

October 2, 2000

by

Peter Christiansen, NBI, pchristi@nbi.dk

Introduction

The purpose of this note is to explain the clustering algorithm.
First a fast overview will be given about the code and how to use it. The main methods of the classes will then be explained in detail with some comments about what might be done differently.

Overview

The cluster algorithm involves the modules : **BrClusterFinderBase**, **BrTPCClusterFinder** and **BrTPCHitClusterFinder**. The input data is a table of **BrTPCSequence**s that are grouped into **BrTPCCluster**s. The clusters are tested for size and shape and if needed, deconvoluted into smaller clusters, before they are fitted and stored as **BrTPCHitCluster**s in a **BrTPCHitClusterTable**. The hit clusters are converted to local **BrDetectorHit**s that are in the TPC local tracking and TPM1 vertex finding.

BrClusterFinderBase

This is the base class of **BrTPCClusterFinder** and should also be the base class of alternative cluster finders if such come around. The central piece is the *Event* method which should **NEVER** be overwritten. When the user calls the **BrTPCClusterFinder** *Event* method, this method is used.
First it finds the **BrDataTable** with the **BrTPCSequence**s in the input **BrEventNode**.
Then it calls the *Entry* method that is overwritten by the **BrTPCClusterFinder**.
After the sequences have been grouped into **BrTPCHitCluster**s it performs the final conversion to **BrDetectorHit**s.

BrTPCClusterFinder

When the *Entry* method is called by BrClusterFinderBase it first makes sure the table is sorted so that the positions i, j of two **BrTPCSequence**s in the table obeys :
$i < j$ in table $\Rightarrow$ tpc1 < tpc2
(meaning : row i < row j (if =) pad i < pad j
The sequences are then grouped into **BrTPCCluster**s depending on a "touch" relation. 2 sequences i, j are related if : row i == row j && abs(pad i - pad j) == 1 && timeoverlap > 1 time segment (this value is currently hardcoded.
The sorting limits the loop to only looking in the same row (This could be made even smaller

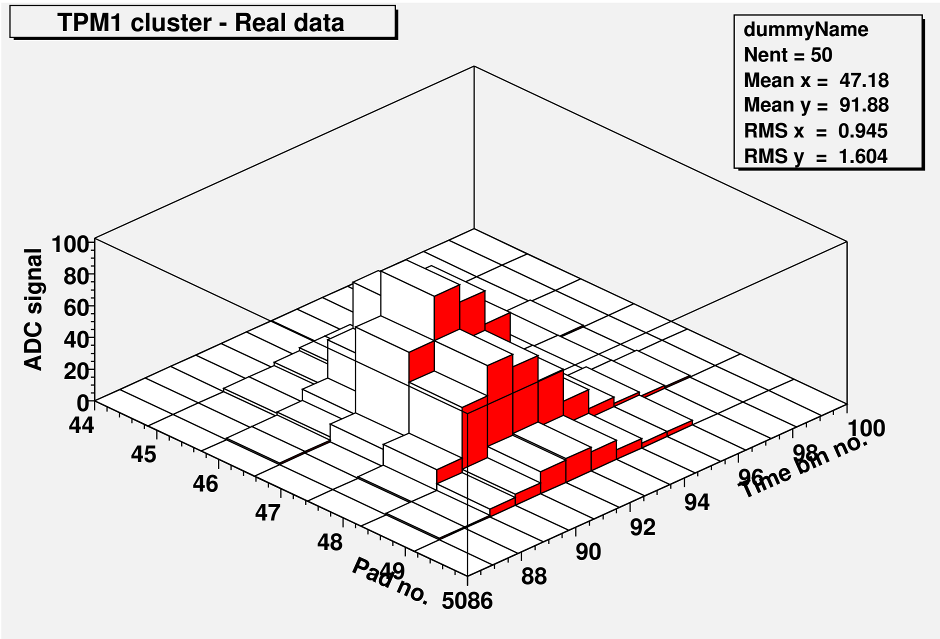

Figure 1: Five TPCsegments grouped into a cluster with the cluster finder.

by limiting to only the next pad).
A cluster should in the best cases look like figure 1.
The list of clusters is applied cuts, demanding more than 1 sequence in a cluster and that the the highest ADC signal is greater than *fMAXPixelCutOff*). The clusters are then divided into 3 categories depending on their shape, psig (pad sigma) and tsig (time sigma) (this is part of the *DeconvoluteCluster* method).
These claculations are now done in BrTPCCluster.
If either psig og tsig is small, the cluster is deemed a noise hit, and is removed.
If either psig or tsig is big, then the cluster is deemed a multicluster and is deconvoluted. Hits with both psig and tsig inbetween the limits are deemed single hits. Finally the **BrTPCCluster**s are passed on to the **BrTPCHitClusterFinder**.

BrTPCHitClusterFinder

The pad projection and time projection of each cluster can be fitted with 3 different functions. A gaussian, a weighted mean, and a time profile, presumably used by STAR, and looking at the real data should probably be used. The only method available at the moment is the weighted mean for both pad and time.
This will be looked into.
One should note that the weigthed mean estimate is the best estimate for a one dimensional gaussian. The position is therefore correct, but additional information such as χ^2 differs. The information from and about the fit is stored in **BrTPCHitCluster**s.

BrTPCSequence

This is the base class for the data read out from the TPCs. The class contains part of the readout from one pad, in principle only a continous string of positive ADC values. The class contains the row number, pad number, the first time bin, the number of time bins in the sequence and a pointer to the ADC values. The content of the class is illustrated in figure 2.

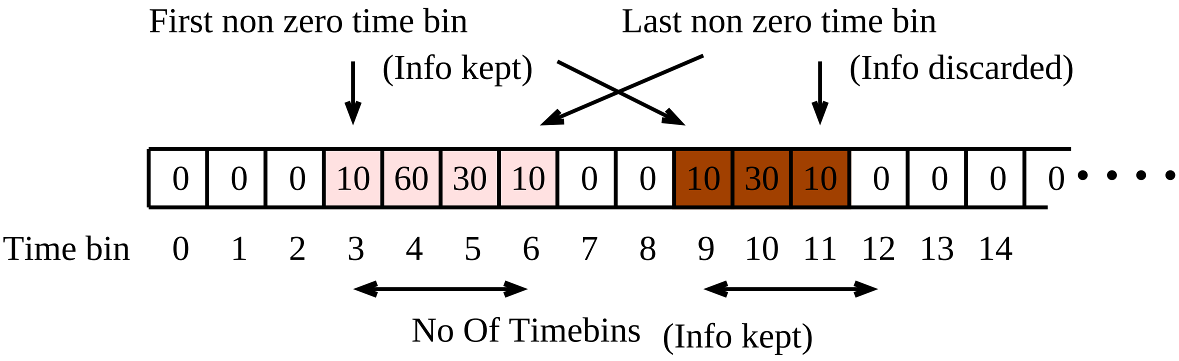

Figure 2: Two **ideal** TPC sequences. For a given row and pad only the segment with non zero ADC values are kept.

The algorithm for real data is that if two or more consecutive timebins have ADC values above threshold + cut. Data 1 time bin before and 1 time bin after are also kept. This can have the effect of seing zero and negative values in the sequences. These can reduce the efficiency of the clustering and should be removed. You can use **BrTPCPreProcess** to strip the sequences of these values.

Example : You have a **BrEventNode*** *event* with the **BrDataTable** *"TPCsequence TPM1"* that contains the raw **TPM1** sequences. To remove zero and negative ADC values do :

```
BrTPCPreProcess *prePro =
   new BrTPCPreProcess( ''TPM1'', ''TPM1 Pre Processor'' );
prePro->SetCutTime( 0 );
prePro->Event( event );
```

BrTPCCluster

The cluster mainly contains a **TList** with pointers to the **BrTPCSequence**s that are part of the cluster. It knows most trivial thigs about the cluster and contains 5 different categorizations of clusters.

Name	fStatus	Meaning
kNoiseHit	3	noise cluster ⇒ remove
kSingleHit	2	good cluster ⇒ ok
kNoiseHit	1	multi cluster ⇒ deconvolute
kSingleHitDec	4	good cluster after deconvolute
kMultiHitDec	5	big single cluster after deconvolute

BrTPCHitCluster

The **BrTPCHitCluster** contains a summary of the fit information, and a pointer to the cluster from which it was derived.

BrDetectorHit

This class is currently only used by the TPC modules. It is general and could perhaps be used for other detectors, DC etc.İt contains a unique Id, the hit position in local coordinates width the estimated uncertainty on only x and y, since usually z is limited by detector design. An integer that can be used for fx. row number, and an integer that can be used to indicate how many times (or if) the hit was used in fx. tracking and an integer that can describe the quality of the hit. I would like to add a general **TObject** pointer to the underlying hit.

How to use

You should only worry about the **BrTPCClusterFinder** class and ignore the rest. The class knows how to deal with the other module classes. It is used in vertex finding (TPM1) and local tracking (all four TPCs). If you want to do it the hard way do :

You still have a **BrEventNode*** *event* with the **BrDataTable** *"TPCSequence TPM1"* that contains the raw **TPM1** sequences.

```
BrTPCPreProcess *prePro =
   new BrTPCPreProcess( ‘‘TPM1’’, ‘‘TPM1 PreProcessor’’ );
prePro->SetCutTime( 0 );
BrTPCClusterFinder *tpm1Cluster =
   new BrTPCClusterFinder( ‘‘TPM1’’, ‘‘TPM1 ClusterFinder’’ );
tpm1Cluster->SetMaxPixelCutoff( 10 );
BrTrackFollowFinder *tpm1Tracker =
   new BrTrackFollowFinder( ‘‘TPM1’’, ‘‘TPM1 tracker’’ );
tpm1Tracker->SetTrackingMethod( 1 );
tpm1Tracker->SetSearchWidthX( 0.2 );
tpm1Tracker->SetSearchWidthY( 0.1 );
prePro->Event( event );
fClusterFinder->Event( event );
BrClonesArray *clusterArray = fClusterFinder->GetDetectorHits();
tpm1Tracker->SetDetectorHits( clusterArray );
BrEventNode *output =
   new BrEventNode( ‘‘TPM1 Node’’ );
tpm1Tracker->Event( in, out );
event->AddEventNode( out );
```

Note that the **BrTrackFollowFinder** with this method needs a geometry file, for the size of the TPC, so in your main program you need something like this :

```
BrGeometryDbManager *geom =
   BrGeometryDbManager::Instance();
geom->SetDbFileName( "90deg.geo" );
```

You can of course also use the **BrTrackStringFinder**. Look at :

`http://lynx.uio.no/trine/brahms/BrTrackStringFinder_doc.html`

to see how to use that one.
You can also do it a little more laid back.

```
BrModuleContainer fContainer =
   new BrModuleContainer( "ALLTPCTracking", "Tracking Container" );

//Add cluster modules to container
Char_t *name[ 4 ] = { "T1", "T2", "TPM1", "TPM2" };
for( Int_t i = 0; i < 4; i++ )
{
   BrTrackFollowFinderModule *tracker
      = new BrTrackFollowFinderModule( name[ i ], "TPC Tracker" );
  // preprocessor stuff
   BrTPCPreProcessor *tpcPreProcess = fTrackModule->GetPreProcessor();
  //  don't cut on time
   tpcPreProcess->SetCutTime( 0 );
   tracker->GetClusterFinder()->SetMaxPixelCutoff( 10.0 );
  // Set tracking method
   BrTrackFollowFinder *trackDummy =
     (BrTrackFollowFinder*)tracker->GetTrackFinder();
   trackDummy->SetTrackingMethod( 1 );
   trackDummy->SetHitSearchWidthX( 0.4 );
   trackDummy->SetHitSearchWidthY( 0.2 );
   fContainer->AddModule( tracker );
}
fContainer->SetVerbose( 1 );
fContainer->ListModules();

//To do tracking when you have an event fEvent :
 fContainer->Event( fEvent );
```

Now you are ready for trying it out. If you have problems ask your local **BRAT** specialist or write me.

The options

BrTPCClusterFinder options :

- **HighCutPsig, HighCutTsig, LowCutPsig, LowCutTsig** cuts on the cluster variances. Default values are 1.6, 1.6, 0.55, 0.55. The default values for time have problems and the high cut should propablu be around 2 for real data. Look at the ghistograms for more detail.
- **fMaxPixelCutoff** cut on the max ADC value for the cluster. Default value is 10.

- **fTestOutputOn** tests the output from the module, i. e. clusters don't have only 1 pad, 1 time bin, to low ADC values, and that the output is in general consistent.

BrTPCHitClusterFinder options :

- **fFindHitsVersion** Differentiates between different versions of FindHits methods. 2 is default meaning method *FindHits2(WEIGHTED_MEAN, WEIGHTED_MEAN)* is used.
- **NB!** It is currently only possible to use the weighted mean method.

The details

Here I will only focus on **BrTPCClusterFinder**. **BrClusterFinderBase** is really simple but there should be a few words on **BrTPCHitClusterFinder**, sorry.

Grouping the sequences into clusters

This is done in a loop over the sorted segments.
The process is illustrated in figure 3. Notice that in step 2 cluster 1 is deleted and the members are inserted into cluster 1.

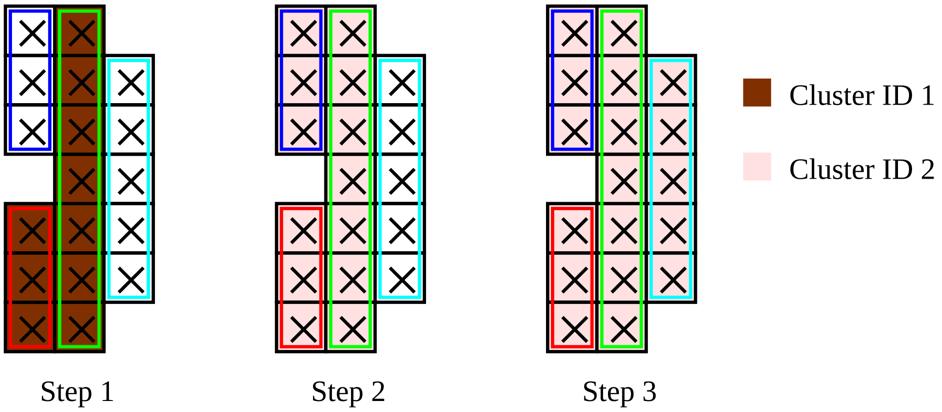

Figure 3: The TPCsegments are grouped in clusters.

Calculating tsig and psig

When the clustering is done the position, width and energy of the cluster is calculated. This is currently done in **BrTPCClusterFinder**.

$$\begin{aligned}
\text{pav} &= \frac{\sum \text{pad\#} \times \text{adc}}{\sum \text{adc}} \\
\text{psig} &= \frac{\sum (\text{pad\#})^2 \times \text{adc}}{\sum \text{adc}} - \text{pav}^2 \\
\text{tav} &= \frac{\sum \text{time\#} \times \text{adc}}{\sum \text{adc}} \\
\text{tsig} &= \frac{\sum (\text{time\#})^2 \times \text{adc}}{\sum \text{adc}} - \text{tav}^2 \\
\text{dEdx} &= \sum \text{adc}
\end{aligned}$$

The clusters are now divided into three groups according to their width (psig, tsig).

$$\text{fStatus} = \begin{cases} \text{kNoiseHit} & \text{if (psig} < \text{psig\_lowlimit)} \vee \text{(tsig} < \text{tsig\_lowlimit)} \\ \text{kSingleHit} & \text{else if (psig} < \text{psig\_highlimit)} \wedge \text{(tsig} < \text{tsig\_highlimit)} \\ \text{kMultiHit} & \text{else} \end{cases}$$

Deconvolution

The clusters with fStatus = kMultiHit have to be deconvoluted. The approach taken in the code is very simple at the moment. The cluster is divided into subclusters with the algorithm shown in figure 4. Then pav, psig, tav, etc. is calculated for the subcluster and the noise test of the earlier section is applied. Good hits from deconvoluted clusters are assigned fStatus = kSingleHitDec and hits that are still to big are assigned the value fStatus = kMultiHitDec and are not removed.

Improvements : Figure 5 shows an example of how the algorithm deconvolutes a multi cluster. In this situation the identification of the 2 subclusters is not very good. It would be better to fit the cluster distribution with some parameters such as the width fixed or some relative parameter fixed. As such I think the peak finding part of the algorithm is very useful, since it gives an idea avout how many subcluster are needed and where there peaks should lie.

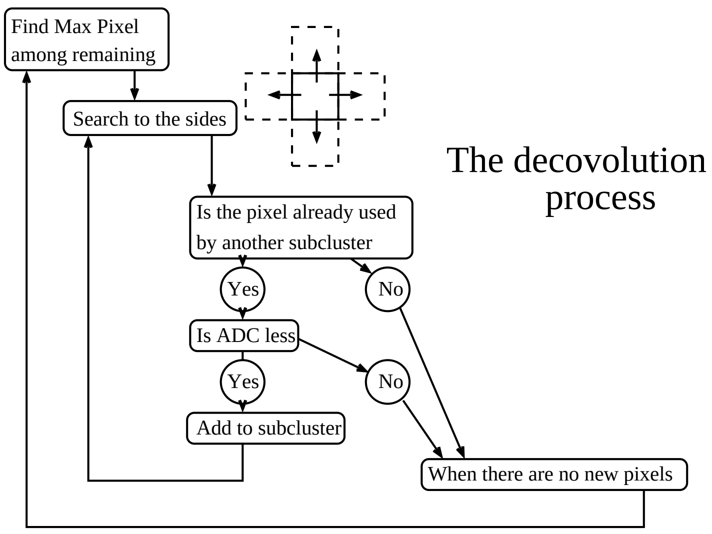

Figure 4: Deconvolution. Search to the side of a pixel means comparing to (pad+1, time), (pad-1, time), (pad, time+1), and (pad, time-1)

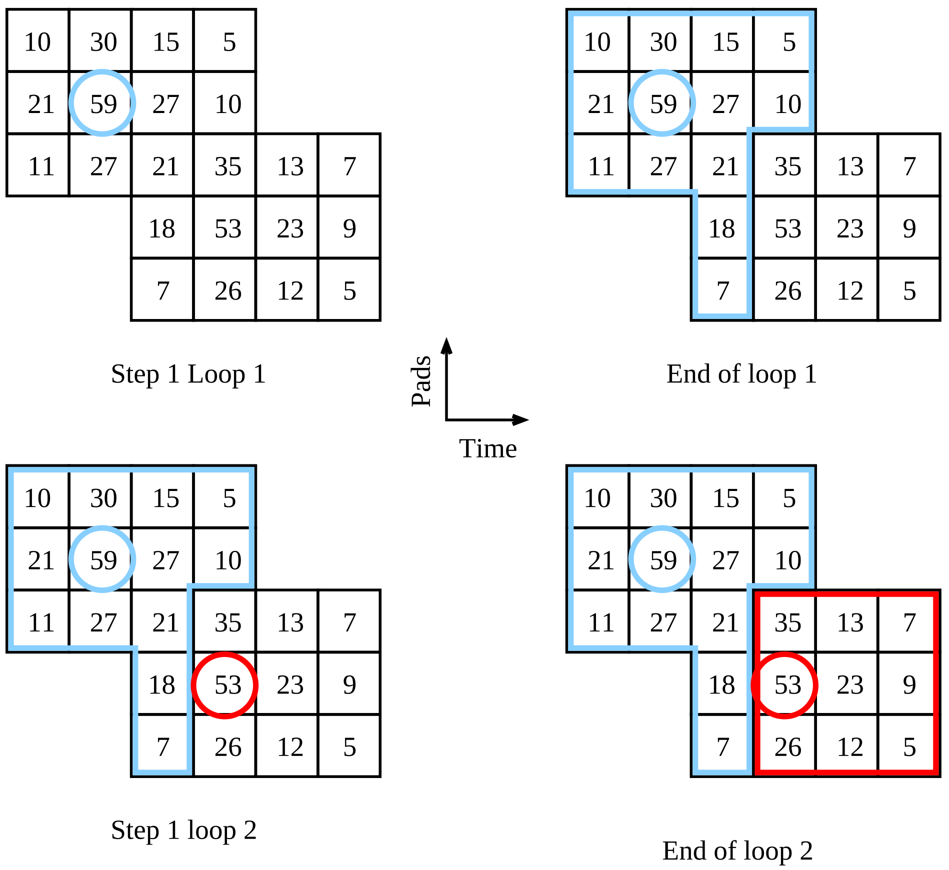

Figure 5: An example of how the deconvolution algorithm works.