Algorithms for online vertex reconstruction at CMS

Michael Hutcheon

Summer 2016

Abstract

Three vertex reconstruction algorithms are investigated for hardware implementation in the proposed 2025 upgrade for the CMS detector at the high-luminosity LHC (HL-LHC). The conditions at the HL-LHC are simulated with a mean of 150 minimum bias events per proton bunch crossing (pileup) at a centre of mass energy of 13TeV. Proposed online track reconstruction performance is also modelled using a P_T cuttoff of 3GeV ($\equiv P_T^{min}$) on tracks and by applying a Gaussian smearing with $\sigma_{track} = 0.25$mm to the track positions (these are later varied); we denote vertices that still leave a tracking signature after these cuts as 'reconstructable'. Several aspects of vertex reconstruction algorithm performance are investigated. Under HL-LHC conditions it is found to be possible to reconstruct the number of vertices with a standard deviation of 1.384 vertices out of a possible 12.4 (on average) reconstructable vertices in a sample of 1000 events ($\sim$ 150,000 individual minimum bias *pp* interactions). It is found to be possible to reconstruct their position with a standard deviation of 0.9211mm and to obtain an average of $>$ 90% correct track-vertex association for a $t\bar{t}$ production vertex. we also investigate the effects of varying the mean piluep (up to a maximum pileup of 1000 minimum bias *pp* interactions per event) and the behaviour of the prerequisite track reconstruction is modified (by varying P_T^{min} and σ_{track}). The possibility of reconstructing total pileup energy from this reconstructed vertex information for use in jet subtraction is investigated, and found to be unlikely.

Contents

1 Introduction

Data from the LHC is typically treated in two stages; 'online' and 'offline', the online stage takes place at CERN and the offline stage at large computing centres across the globe. Currently CMS operates a 2-level trigger system online in order to remove less interesting events; level-1 is at the detector stage, implemented in custom-made electronics; level-2 is reffered to as the high-level trigger (HLT) and runs streamlined versions of the offline reconstruction algorithms on a processor farm at CERN.

The upgrade of the LHC to the High Luminosity LHC (HL-LHC) increases the amount of raw data produced in collisions to beyond the capabilities of the current CMS detector and data processing at CERN. In order to cope with the increased collision rate at the HL-LHC more aggressive data cuts must be made at the level-1 stage, before the data leaves the detector. Currently these level-1 data cuts are based mainly on information from calorimetry, with events being kept on the merit of their hardronic and leptonic energy spectra. However, with the proposed upgrades to CMS we will for the first time have access to tracking information online, at the hardware level; this provides an opportunity to incorporate some of the higher level reconstruction algorithms into this stage, with a view to performing more useful data cuts.

With the increased luminosity each collision event also becomes more complex, with doezens of soft QCD proton-proton interactions ('pileup') accompanying interesting hard QCD interactions. Many of the more useful event reconstruction algorithms used offline and at the HLT require accurate information on the location and number of these pp interaction vertices (in Jet reconstruction and pileup subtraction for example) as well as information on track-vertex association. However, the current offline algorithms used to recover this information are too complex to implement in online hardware. A useful question to ask is then: How well can we reconstruct these vertices at the online stage, in hardware? In an attempt to shed some light on this question this study analyses the performance characteristics of a variety of simple vertex reconstruction algorithms, under the simulated conditions of the High Luminocity LHC.

2 Simulation

2.1 Simulating collision events

In order to analyse the expected performance of possible reconstruction algorithms we must simulate the events that we expect to see at the HL-LHC. In order to do this we use the monte carlo event generator PYTHIA (version 8.219) and the detector response simulation framework Delphes (version 3.3.2). Our samples consist of a single $gg \to t\bar{t}$ production event combined with an average of 150 minimum bias pileup interactions. These interactions are distributed along the beamline (the z direction) in a Gaussian distribution with a standard deviation of 5cm, the number of pileup interactions is modelled as a Poisson distribution with a mean of 150. A $t\bar{t}$ event was included in each sample in order to assess the performance of the algorithms in the reconstruction of hard event verticies as well as minimum bias verticies. PYTHIA was used to generate the $t\bar{t}$ and minimum bias events individually and Delphes was used to superimpose these events onto one another to form the final sample event. The PYTHIA and Delphes card files used are included in the appendix.

Note that for this study, secondary production vertices are included as well as primary proton-proton vertices in order to assess the robustness of the algorithms against

the presence of these secondary production vertices. It is not uncommon to have decay series including relatively long-lived particles such that we see tracks displaced from the proton-proton interaction point by several tens of mm; included in the appendix is a decay chain diagram for the entirety of a minimum bias event, with the z coordinates of highly displaced particles shown (and highlighted) to illustrate this effect; this was not a hand-picked event, it was simply the first one generated.

2.2 Simulating track reconstruction

After the pileup interactions have been superimposed upon the $t\bar{t}$ event we then perform cuts and smearing formulae to the resulting particle tracks in order to model the proposed new features of CMS. These cuts are intended to model the track reconstruction algorithm performance that will be available in the upgraded CMS detector, as this is the input to the vertex reconstruction algorithm. As the system is still in the design phase it is unknown what the exact performance characteristics of the track reconstruction will be. To this end we model a range of possible scenarios, with a view to both analysing possible vertex reconstruction performance, and to providing information on how much there is to gain on this front if the track reconstuction performance is improved. The proposed track reconstruction system exhibits a limited resolution of the track beamline crossings; we model this as a Gaussian smearing of the crossing position, we use widths ranging from 100 μm to 1mm. On top of this, currently tracks can only be reconstructed down to a transverse momentum (P_T) of 3 GeV (compare this to approx 100 MeV for offline analysis) so we also investigate the effects of removing tracks below various P_T thresholds (P_T cuts); we denote the minimum P_T at which tracks can be reconstructed as P_T^{min}. For later parts of the study involving actually running the reconstruction algorithms we also apply a P_T cut of 100MeV (to account for tracks wound up by the magnetic field and lost) and and η cut of 5 (to account for the geometry of the tracker) to the truth information before we do anything else, as we are not concerned with these undetectable tracks.

2.3 Simulation results

The results of the event and detector response simulations are summarized in this section. These results serve as input into the vertex reconstruction algorithms that we wish to investigate. We are most interested in reconstruction of vertex position in the z coordinate (along the beam direction) as the spread of the vertices in the x and y directions is comparatively much smaller than that in the z direction for a given proton bunch crossing. Figure 1 is an example of the z distribution of the points of nearest approach of tracks to the beamline with a Gaussian smearing to model detector resolution, figure 2 is the pp interaction vertices that produced these tracks. These diagrams only show the region $|z| < 10mm$ in the interest of clarity, where $z = 0$ represents the nominal origin (i.e the centre of the pileup distribution); note that this is the region of densest pileup due to the Gaussian nature of the pileup distribution we are using. At this stage no cuts have been applied to the tracks used to create these plots, these come later when we begin to model detector acceptance.

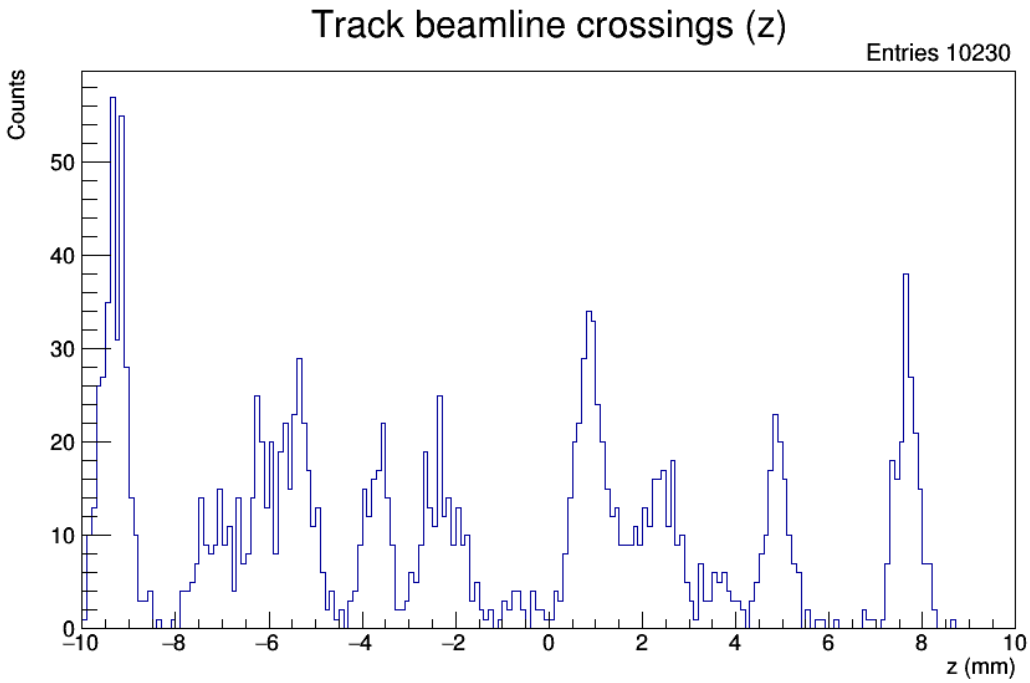

Figure 1: z distribution of track start points (bin size 0.1 mm, no P_T cut, smeared with $\sigma = 0.25mm$)

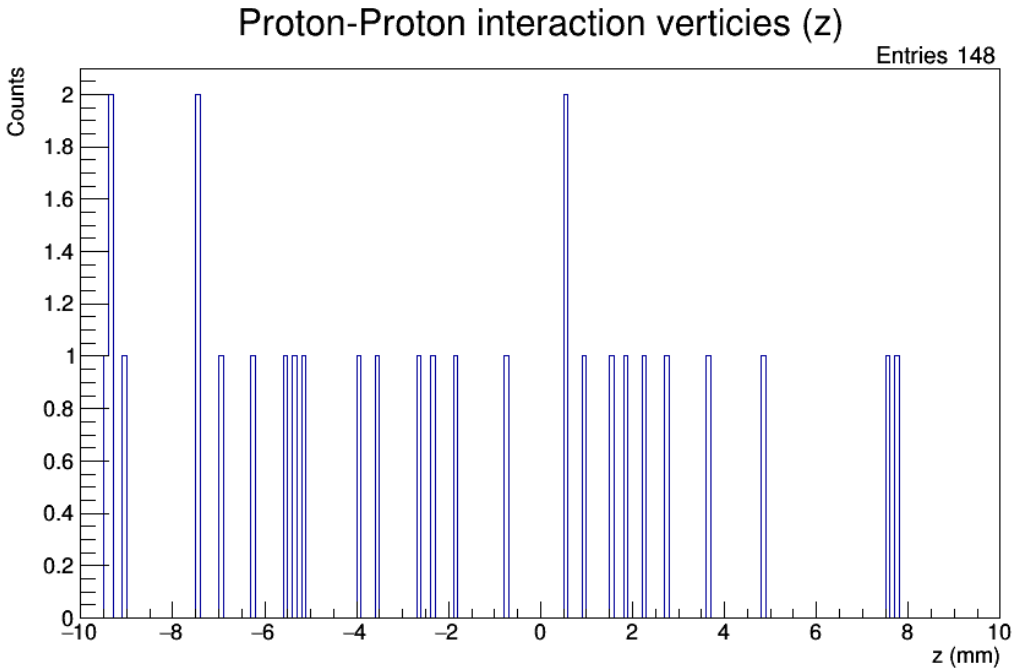

Figure 2: z distribution of proton-proton interaction verticies (bin size 0.1 mm)

A P_T cut is then applied to the tracks from figure 1 in order to finally recover the tracks that we have access to in hardware. The remaining tracks are the input for the vertex reconstruction algorithm (we denote these 'reconstruction points') and are shown in figure 3 for $P_T^{min} = 3$ GeV.

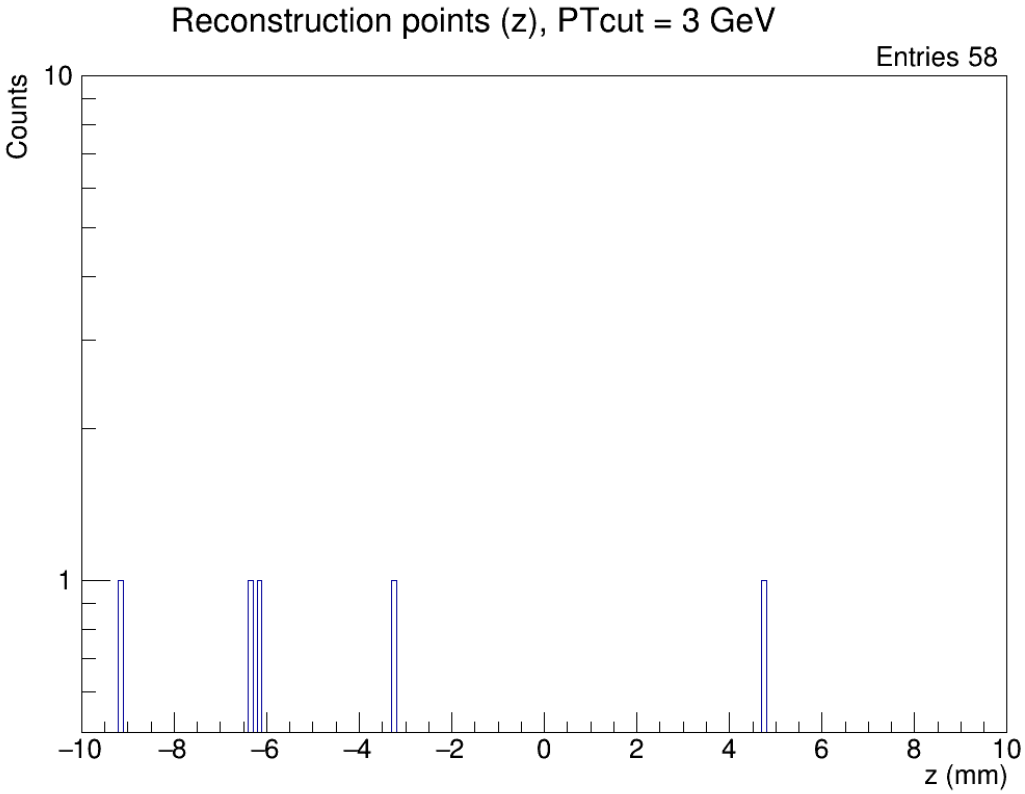

Figure 3: z distribution of reconstruction points (bin size 0.1 mm, $P_T^{min} = 3$ GeV, smeared with $\sigma = 0.25mm$)

The P_T cut decimates the number of tracks by a huge factor. This is to be expected because the vast majority of tracks have relatively low P_T; this can be easily seen by simply plotting a P_T distribution for minimum bias events. This distribution is shown in figure 4 (Note the logarithmic axis).

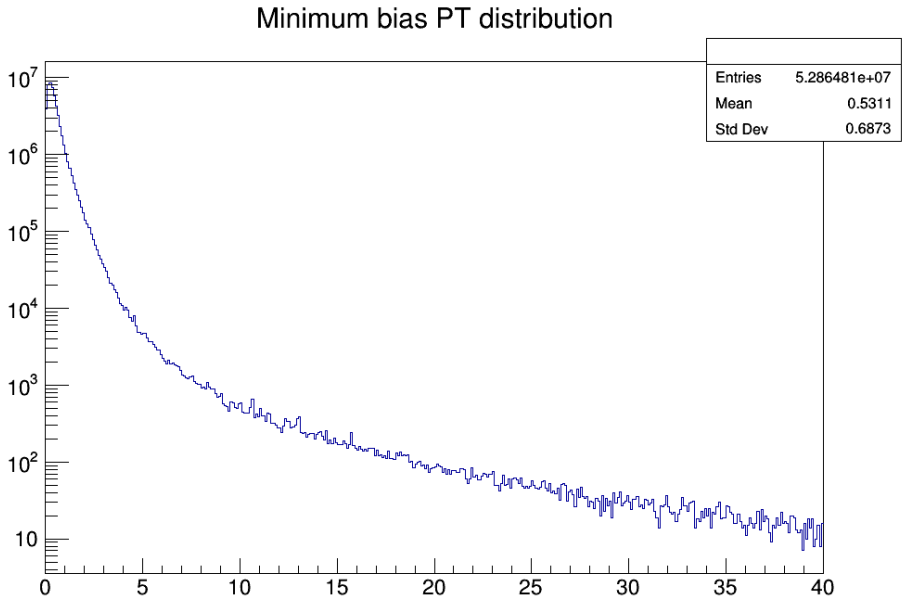

Figure 4: Minimum bias track P_T distribution (From $\sim$ 750,000 minimum bias pp collisions (5000 events, $\sim$ 150 pileup each))

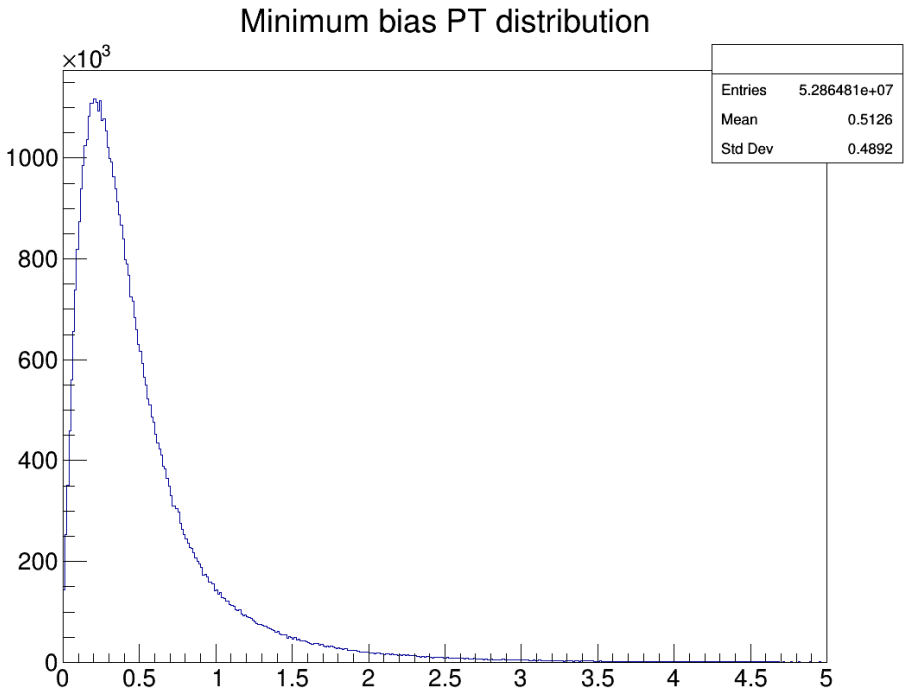

Figure 5: Same as figure 4, rescaled (no log scale)

We can see from figures 4 and 5 that we lose around three orders of magnitude in the number of tracks after a P_T cut of 3 GeV, and that the number of tracks falls off very quickly with P_T. In an ideal world we would attempt to reconstruct the contents

of figure 2, but given that our input is the contents of figure 3; we see that this is an unrealistic goal. However, the remaining high P_T tracks are the most interesting from the point of view of identifying interesting physics. Thus we set as the goal of our algorithm to reconstruct the vertices that produce tracks above a given P_T threshold, P_T^{min} , which is imposed by the track reconstruction that came before. Currently $P_T^{min} =$ 3GeV is the best estimate of how these algorithms will perform.

3 Algorithms

The vertex reconstruction algorithms that we are interested in for this study are designed to take a list of z positions that represent each track as input and produce a list of the optimal assignment of vertices to these tracks. No information is used to arrive at these vertex positions other than these start points; the algorithms are purely geometric. We give the algorithms the z coordinates of the production vertices of particles that produce tracks as input.

Vertex reconstruction algorithms are, to a first approximation, simply specialized versions of clustering algorithms. In this study we consider algorithms to derive from two broad categories, agglomerative and divisive. Divisive algorithms start by considering the complete set of input points as an initial cluster and progress by repeatedly subdividing clusters until the most compatible set is formed. Agglomerative algorithms, conversely, start by considering each individual element as a seperate cluster and combining the most compatible clusters at each stage. We implement the algorithms in C++ code. This section provides descriptions of the algorithms used, in increasing order of sophistication. The analysis of the performance of these algorithms is in section 4.

3.1 Gap clustering

One of the simplest approaches to this problem is to apply a gap clustering algorithm. This algorithm is a divisive algorithm which splits the groups of tracks into clusters wherever a sufficiently large gap between tracks occur. The gap clustering algorithm was used in the early days of the LHC, before pileup became an issue. Ironically, this algorithm finds merit here in high pileup conditions due to the comparatively few numbers of tracks remaining after the P_T cut imposed by the online track reconstruction algorithm. The algorithm is outlined in the following pseudo-code implementation:

```
gapCluster(trackList, splitDistance)
{
    vertices = empty set
    cluster = empty set

    sort trackList in ascending z

    foreach (track t in trackList) in order
    {
        add t to cluster
        if (distance from t to next track in trackList > splitDistance)
        or (t is the last track)
        {
            vertex = create vertex from cluster
            add vertex to vertices
            empty cluster of tracks
        }
    }
    return vertices
}
```

The algorithm requires one external parameter, the gap size, to be specified. Whilst relatively easy to implement for our analysis, this algorithm is unlikely to be easily implemented in hardware in it's above form because it requires the set of tracks to be sorted; a non-trivial task. However, it serves as a good baseline on which to judge other algorithms.

3.2 Simple merge algorithm

Having implemented the simplest form of divisive algorithm, it is instructive to implement an agglomerative algorithm. The implementation that we use requires a metric for determining the compatibility of two algorithm clusters. This metric takes a pair of clusters as input and returns a number that allows us to identify the most compatible pair. An overview of different metrics is provided in [1]. For this study we use the 'minimax' metric, which operates as follows: Let α and β be two clusters, and let $S_{\alpha\beta}$ be the set of all distances between track pairs with one track in α and one track in β; the minimax metric is then $M_{\alpha\beta} = max(S)$, and results in what is known as 'comlpete linkage'. We merge the two clusters with the lowest M score. Below is a pseudo-code implementation of a simple merge clustering algorithm. In the interest of speed on a normal computer we first sort the list of starting clusters into ascending z order in order to make calculation of the minimum $M_{\alpha\beta}$ faster. Implementations without this sort exist, but are slower; typically they involve comparing all possible pairs of clusters, rather than just the nearest-neighbour comparision that the sort allows us to carry out.

```
mergeCluster(trackList, maxMergeScore)
{
    clusters = empty set
    foreach (track t in trackList)
    {
        startCluster = create cluster containing only t
        add startCluster to clusters
    }
    sort clusters in ascending z

    loop forever
    {
        minimumScore = infinity
        mergeCluster1 = null cluster
        mergeCluster2 = null cluster
        foreach (cluster c in clusters) in order
        {
            if (c is last cluster)
                exit foreach loop

            nextCluster = next element in clusters
            M = metric(c, nextCluster)
            if (M < minimumScore)
            {
                minimumScore = M;
                mergeCluster1 = c;
                mergeCluster2 = nextCluster;
            }
        }
        if (minimumScore > maxMergeScore or mergeCluster1 == null cluster)
            exit forever loop

        merge mergeCluster1 into mergeCluster2
        remove mergeCluster1 from clusters
    }
    return clusters
}
```

This algorithm requires one external parameter, maxMergeScore, which specifies when to stop merging clusters. It is the maximum value of $M_{\alpha\beta}$ at which merging clusters α and β is permissible. As previously mentioned, the algorithm also requires a metric to be specified; a pseudo-code implementation of the 'minimax' metric that we use is below:

```
metric_miniMax(cluster1, cluster2)
{
    maxDistance = 0
    foreach (track t1 in cluster 1)
    {
        foreach (track t2 in cluster 2)
        {
            seperation = absolute distance from t1 to t2
            if (seperation > maxDistance)
                maxDistance = seperation
        }
    }
    return maxDistance
}
```

This metric runs in $O(N_\alpha N_\beta)$ time, where N_α and N_β are the numbers of tracks in clusters α and β respectively. We also test a metric that runs in $O(N_\alpha + N_\beta)$ time, based on calculating the standard deviation of the combined track distribution:

```
metric_standardDeviation(cluster1, cluster2)
{
    averagePosition = average position of tracks in cluster1 and cluster2
    standardDeviation = 0

    foreach(track t1 in cluster1)
        add distance from t1 to averagePosition to standardDeviation

    foreach(track t2 in cluster2)
        add distance from t2 to averagePosition to standardDeviation

    divide standardDeviation by total number of tracks in cluster1 and cluster2
    return standardDeviation
}
```

This metric produces simmilar results to the miniMax metric.

3.3 DBSCAN algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [2] is an agglomerative algorithm that offers several benefits over the previous two methods. Firstly, it does not require at any stage for the list of tracks to be sorted in any way; greatly improving the chance that it might be possible to implement in hardware. Secondly, it is robust against outliers in that it has the capability to mark a particular track as 'noise'. The algorithm connects groups of tracks that reside within a distance ϵ of one another, requiring that the groups have a minimum number of tracks; this property makes the algorithm well suited to the task of vertex reconstruction as often we require that a sensible vertex has at least two constituent tracks. It also produces a result that is insensitive to the shape of the distribution of tracks within a vertex; this is due to the fact that it is a density based clustering, unlike some distribution-led approaches. Below is a pseudo-code implementation of the DBSCAN algorithm:

```
dbscanCluster(trackList, minTracks, eps)
{
    clusters = empty set

    foreach(track t in trackList)
    {
        if (t has been visited by the algorithm already)
            continue to next track

        mark t as visited
        neighboringTracks = tracks from trackList within distance eps of t

        if (number of tracks in neighboringTracks < minTracks)
            mark t as noize
        else
        {
            c = new cluster
            add t to c
            foreach(track t' in neighboringTracks)
            {
                if (t' is not visited)
                {
                    mark t' as visited
                    neighboringTracks' = tracks within distance eps of t'
                    if (number of tracks in neighboringTracks' >= minTracks)
                        neighboringTracks = neighboringTracks combined with neighboringTracks'
                }

                if (t' is not yet a member of any cluster)
                    add t' to c
            }
            add c to clusters
        }
    }

    return clusters
}
```

This algorithm takes two parameters; minTracks and eps (ϵ). The parameter ϵ specifies the maximum distance between tracks that we may consider to have originated from the same vertex, minTracks specifies the minimum number of tracks we require a vertex to have. These easily interpretable parameters makes the DBSCAN algorithm a strong contender for a vertex reconstruction algorithm.

4 Algorithm Performance

It is a non-trivial task to come up with a reasonable measure of the performance of these algorithms; their success depends heavily on how the resulting vertex reconstructon information is used. As a result we present several different measures of algorithm performance for each of the above algorithms, for the default P_T^{min} of 3GeV and track smearing of $\sigma = 0.25$mm.

4.1 Vertex count reconstruction

The first, and most obvious, measure of performance is simply the number of vertices that we reconstruct compared to the true number of vertices. As mentioned in section 2.3 we treat the true number of vertices as the number of vertices that have at least two tracks with $P_T > P_T^{min}$, as we have no hope of reconstructing vertices if their tracks dont have enough P_T to be reconstructed. All of the above algorithms have the scope to both under-reconstruct or over-reconstruct the vertices; as such we cannot sensibly define a reconstruction efficiency, instead we look at the amount that the algorithm 'missed' the correct number by. Any parameters for the algorithms were chosen so that the average number of reconstructed vertices per event matched as closely as possible to the true number of vertices when averaged over many samples. Figure 6 shows the reconstruction correlation for the gap clustering algorithm (left) and the number of verticies that the algorithm missed by (reconstructed - truth, right). Figure 7 shows the same plots for the merge algorithm under the same conditions, and Figure 8 for the DBSCAN algorithm.

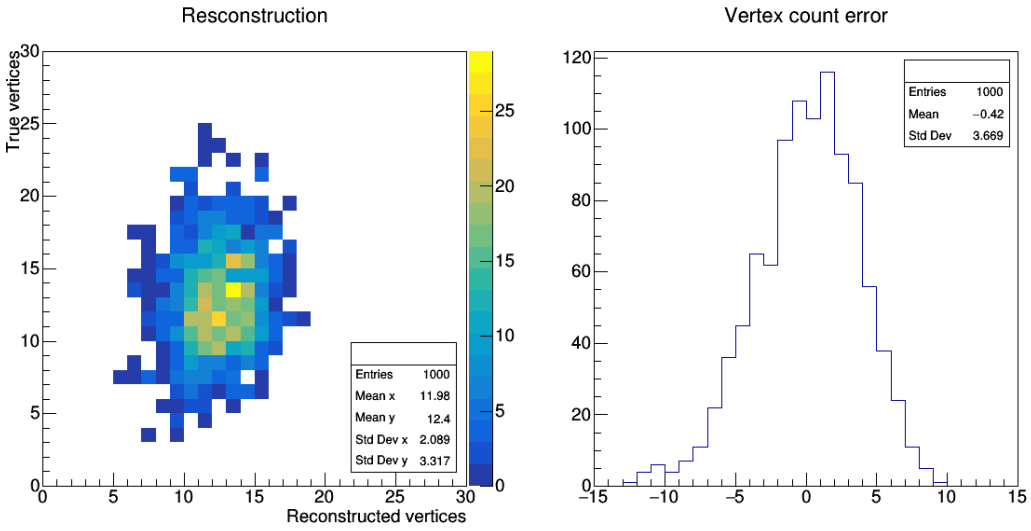

Figure 6: Reconstruction of vertices by the gap clustering algorithm (with gap set to 5.5mm)

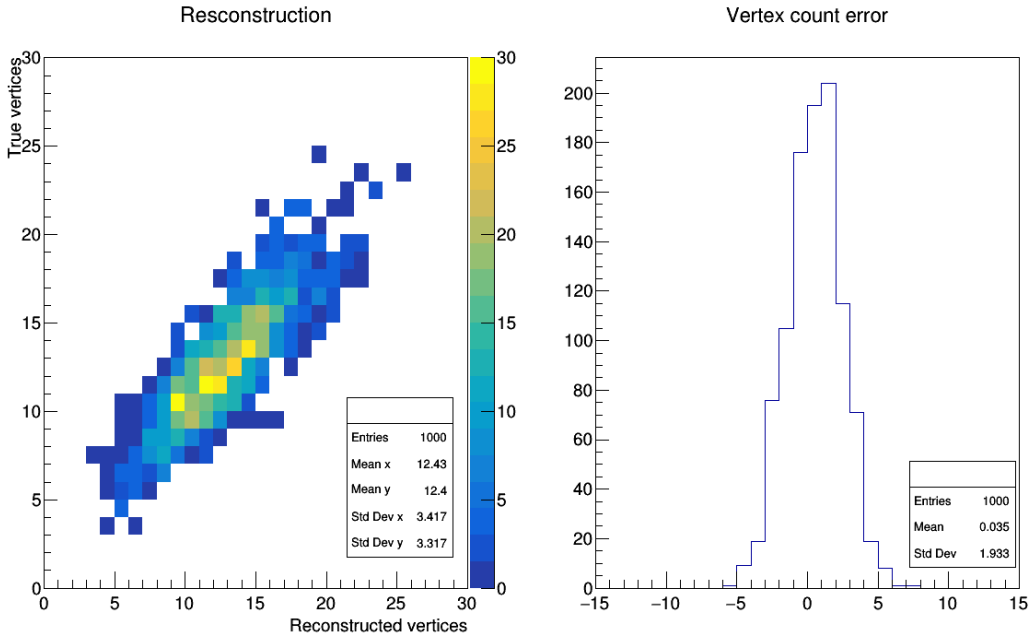

Figure 7: Reconstruction of vertices by the merge clustering algorithm (with merge distance set to 0.325mm, using the 'minimax' metric)

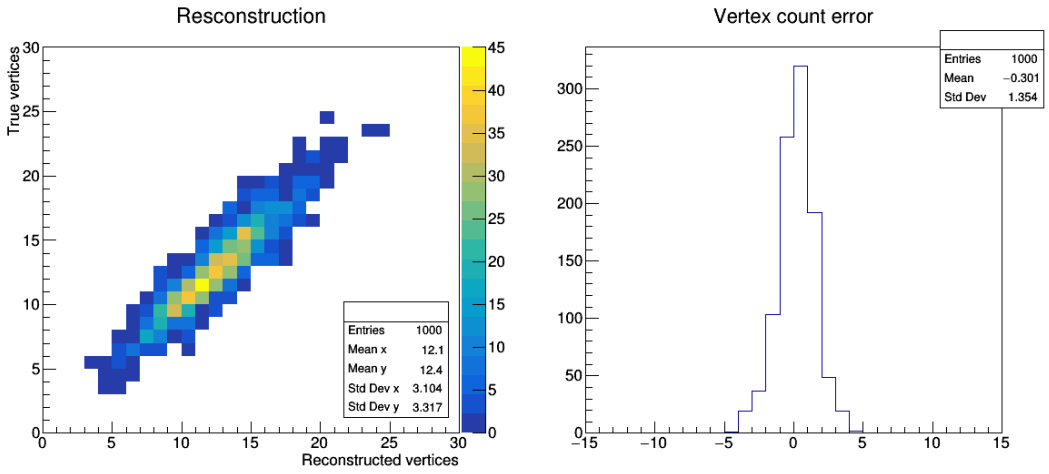

Figure 8: Reconstruction of vertices by the DBSCAN clustering algorithm (with $\epsilon = 0.75$mm and minPoints = 2)

It is clear to see the performance increase from figure 6 to figure 7. The largely uncorrelated blob and wide error distribution in figure 6 show that the gap algorithm is quite a poor choice; largely due to the fact that usually the vertices are far from equally distributed in space, which is when this algorithm performs well. In contrast, the merge algorithm performs much better, as can be seen from figure 7; it shows strong, but not perfect, correlation between the true and reconstructed vertices, as well as significantly reduced errors over the gap algorithm. From figure 8 we can see that the DBSCAN algorithm gives another improvement in performance, perhaps somewhat due to the

algorithm being robust against outliers. The performance of the DBSCAN algorithm also seems to be reasonably insensitive to the parameter ϵ, converging to near optimal performance for a reasonable range of 0.5mm $< \epsilon <$ 1mm; this is in contrast to the gap and merge algorithms which required considerable fine-tuning of parameters to produce the above results.

4.2 Vertex location reconstruction

The next measure of algorithmic performance that we use is the distance between a true vertex and the nearest reconstructed vertex. This serves as a measure of how well an algorithm can reconstruct the location of a particular vertex. Figure 9 shows the distribution of displacements of the true vertex from its nearest reconstruction for the gap algorithm. Figures 10 and 11 show the same for the merge and DBSCAN algorithms respectively. Again these are with $P_T^{min} = 3$GeV and a track smearing with $\sigma = 0.25$mm.

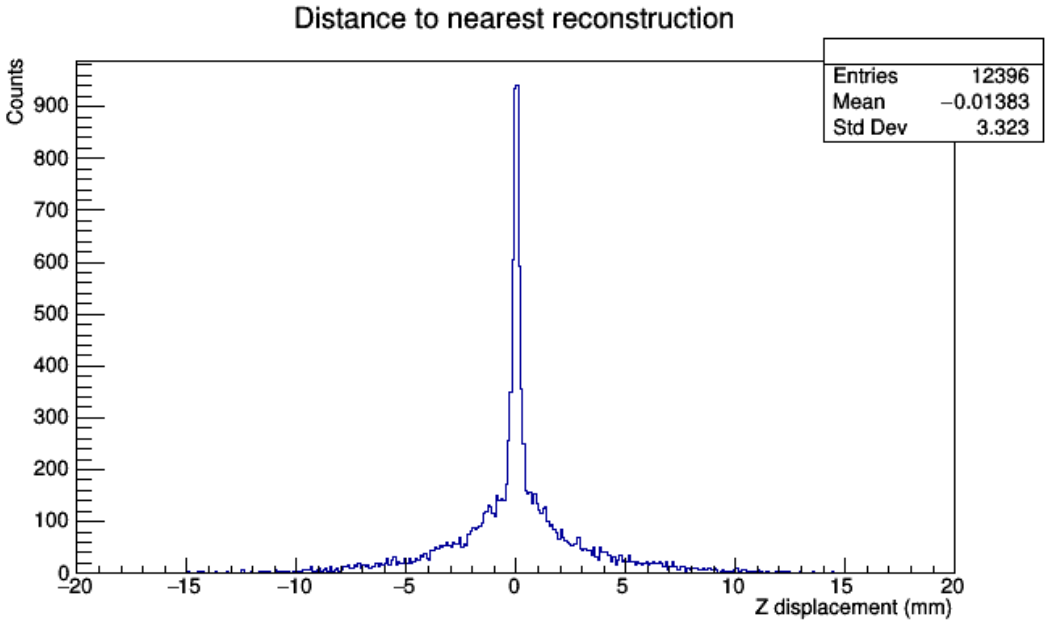

Figure 9: Distances to nearest reconstructed vertex, Gap algorithm (with gap set to 5.5mm)

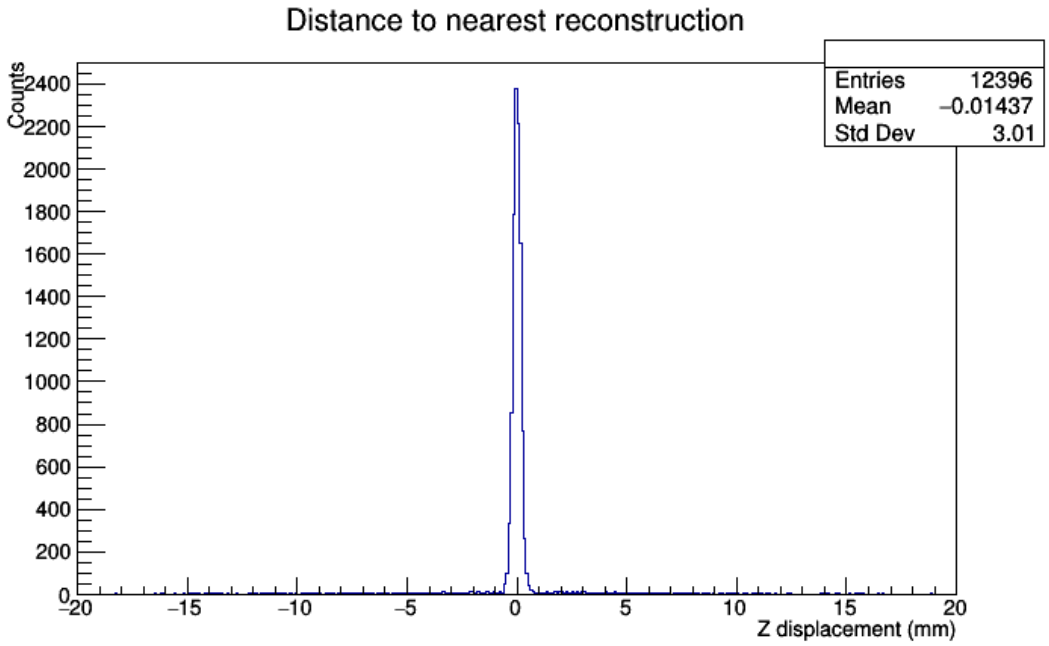

Figure 10: Distances to nearest reconstructed vertex, Merge algorithm (with merge distance set to 0.325mm, using the 'minimax' metric)

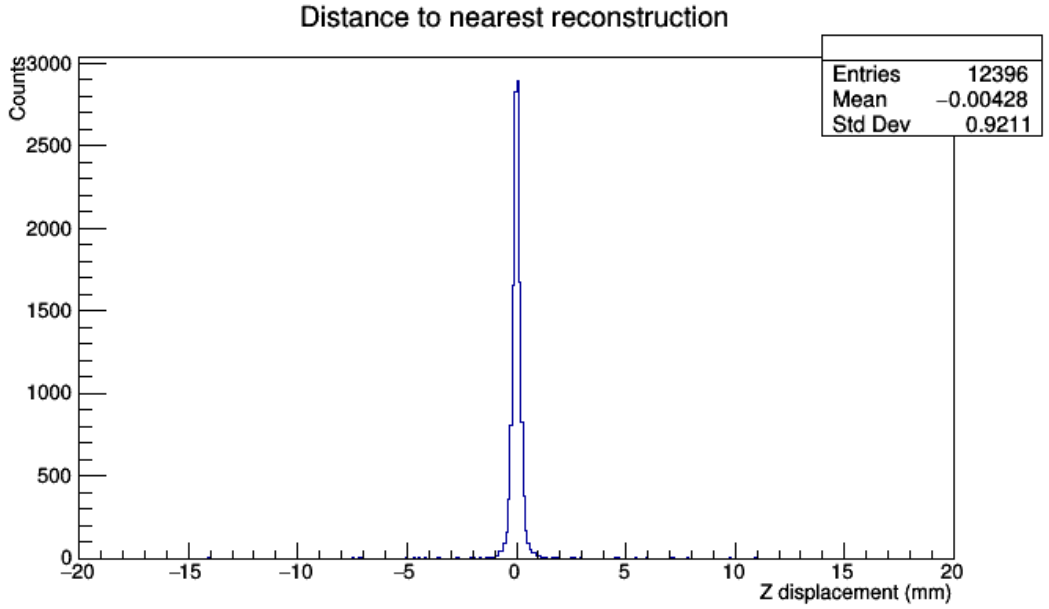

Figure 11: Distances to nearest reconstructed vertex, DBSCAN algorithm (with $\epsilon = 0.75$mm and minPoints = 2)

Again we see that the gap algorithm is a poor choice due to the greatly increased spread of the displacements; this can mostly be put down to the fact that the gap al-

gorithm doesn't generally reconstruct the correct number of vertices (c.f figure 6), so will often not find a suitable match for a given true vertex. The performance increases in the same way as in section 4.1 with the DBSCAN algorithm performing best; this is most easily seen by comparing the standard deviations of plots 10 and 11.

It should be noted that these plots alone are not enough to specify the performance of the algorithms, they should be considered simultaneously with the plots from section 4.1; for example, if an algorithm vastly over-reconstructs the number of vertices, then there is bound to be a reconstruction quite close to one of the true vertices. It is not possible to determine if this, or similar, problems arise from the plots in this section alone.

4.3 Hard event track association

A further measure of the performance of these algorithms (with a slightly more physical motivation) is the ability of an algorithm to correctly associate tracks that came from a hard event to the reconstructed vertex of that hard event. We use the $t\bar{t}$ event included in each sample as an example to test the level of association; we look at the percentage of tracks from the $t\bar{t}$ event that are associated with what the algorithm considers to be the hardest vertex. We append a further step to each algorithm to identify this vertex; we label the 'hardest' vertex as the vertex that maximizes:

$$\sum_{tracks\ n\ in\ vertex} |P_{T,n}| \tag{1}$$

Where $P_{T,n}$ is the P_T of the n^{th} track in the vertex. We then count the tracks from the $t\bar{t}$ event that appear in this vertex in order to calculate the persentage association. This process allows us to examine two aspects of the reconstruction; how well we can associate tracks to a hard vertex, and how well we can identify that hard vertex using equation 1. Figures 12, 13 and 14 show this percentage for the gap, merge and DBSCAN algorithms respectively (note the logarithmic axis).

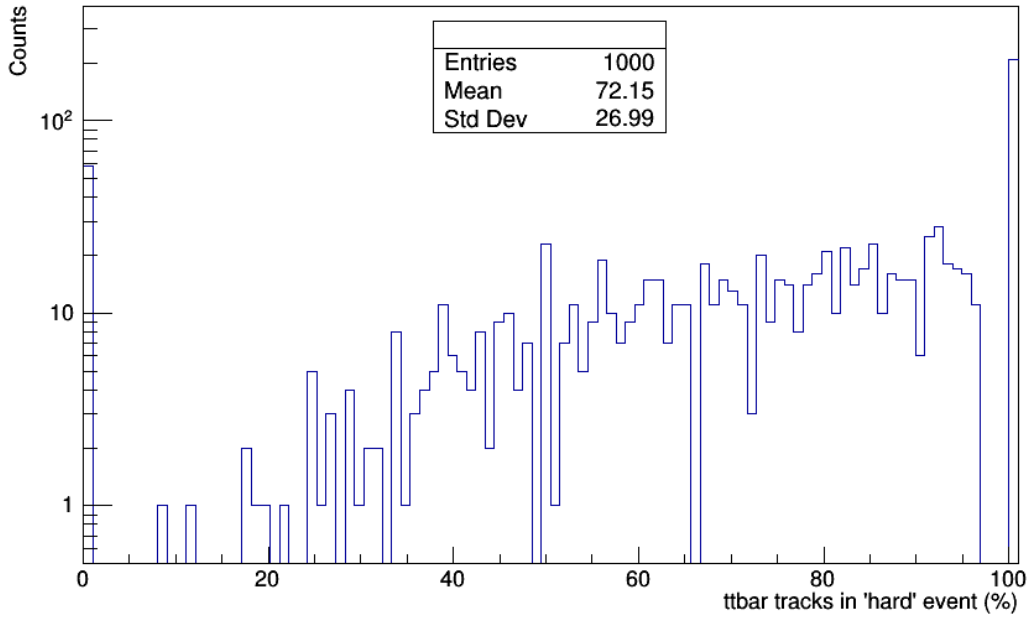

Figure 12: Persentage of $t\bar{t}$ tracks present in hardest vertex, Gap algorithm (with gap set to 5.5mm)

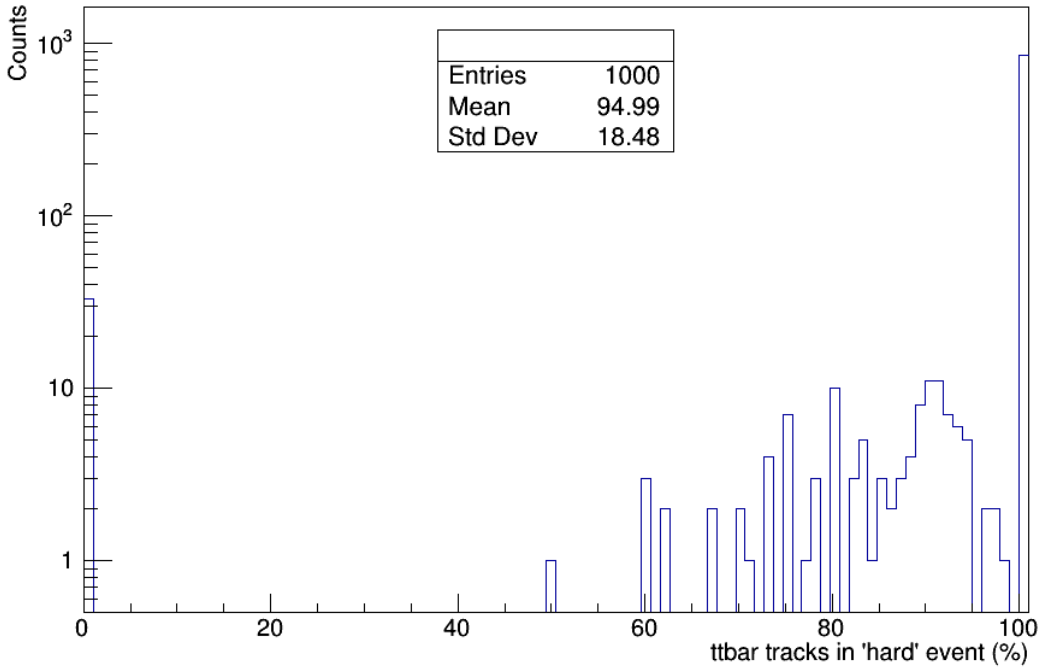

Figure 13: Persentage of $t\bar{t}$ tracks present in hardest vertex, Merge algorithm (with merge distance set to 0.325mm, using the 'minimax' metric)

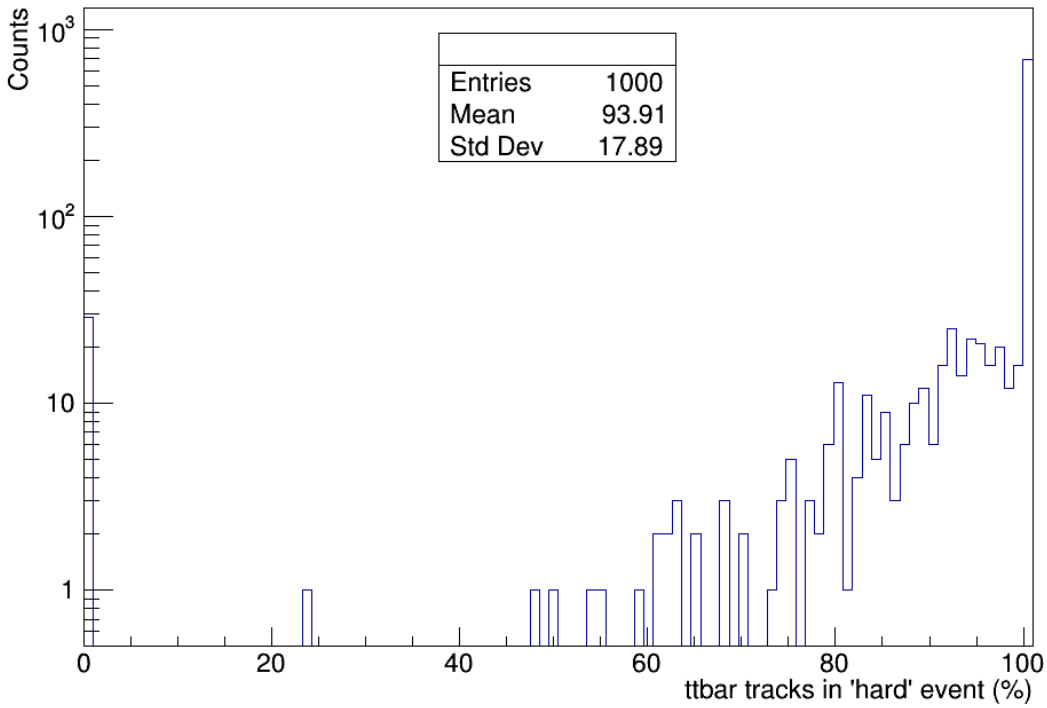

Figure 14: Persentage of $t\bar{t}$ tracks present in hardest vertex, DBSCAN algorithm (with $\epsilon = 0.75$mm and minPoints $= 2$)

It is interesting to note that in these plots the number of entries with a 0% association correspond to the times when equation 1 failed to identify the $t\bar{t}$ event. We also see a significant number of times where the association is perfect, likely corresponding to a well isolated $t\bar{t}$ event. The performance of the algorithm is characterised by how close to 100% the rest of the events lie; these correspond to events where the $t\bar{t}$ event has some number of nearby pileup events with which it may exchange tracks during the reconstruction. By this measure the DBSCAN and merge algorithms perform simmilarily. It may also be interesting to note that none of the algorithms have the capability to 'cross-associate' tracks (see figure 15 below):

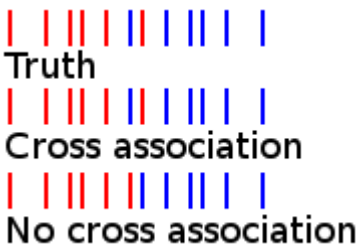

Figure 15: 'cross-association' of tracks to vertices (z axis from left to right, bars represent tracks, colors represent vertex association)

We see that if cross association is impossible, vertices must have well defined edges. This means that as soon as two true vertices are close enough so that their tracks overlap, then some mis-association of tracks must happen; this will be the case with all of the

algorithms in this study. Some possible exceptions would be distribution-led aproaches [3]; these methods are inappropriate here as they require a much more complete set of tracks than we have access to after the P_T cut.

5 Sample variation

5.1 Track reconstruction effects

One of the goals of this study is to analyse how much the accuracy of vertex reconstruction improves given a better, or worse, track reconstruction. So far, the study has focused on modelling the track reconstruction with a position resolution of $\sigma = 0.25$mm and $P_T^{min} = 3$GeV. It is useful to investigate how the performance of these algorithms varies when changing P_T^{min} and σ. The area that these algorithms stand to gain the most in is their estimate of the number of vertices (i.e the results of section 4.1) as they typically perform well in terms of calculating where these vertices are, once found.

5.1.1 Varying P_T^{min}

we have found that the P_T cut has been the dominant factor in reducing the information available (c.f figure 4); so it is interesting to investigate the effects of changing P_T^{min} first. With this in mind, we plot the same reconstruction information from section 4.1 for $P_T^{min} = 2$GeV, again adjusting the parameters to match the mean vertex count. These plots are shown in figures 16, 17 and 18 for the gap, merge and DBSCAN algorithms respectively.

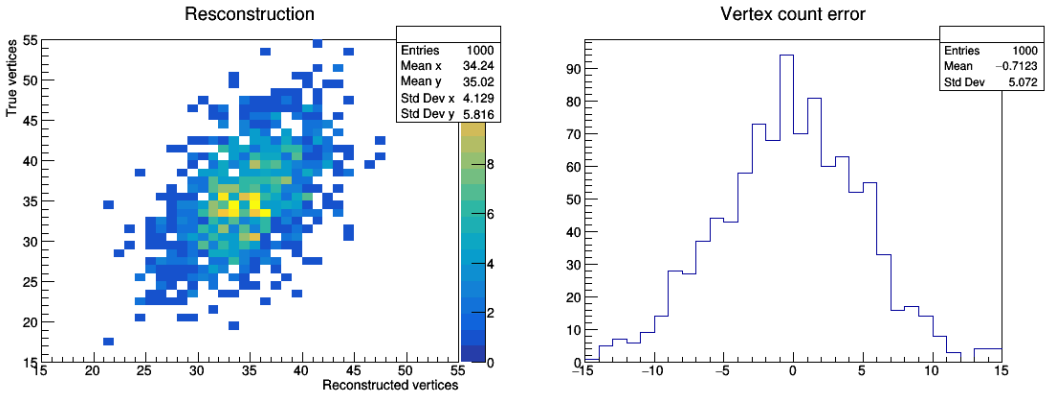

Figure 16: Reconstruction of vertices by the gap clustering algorithm (with gap set to 1.15mm) $P_T^{min} = 2$GeV

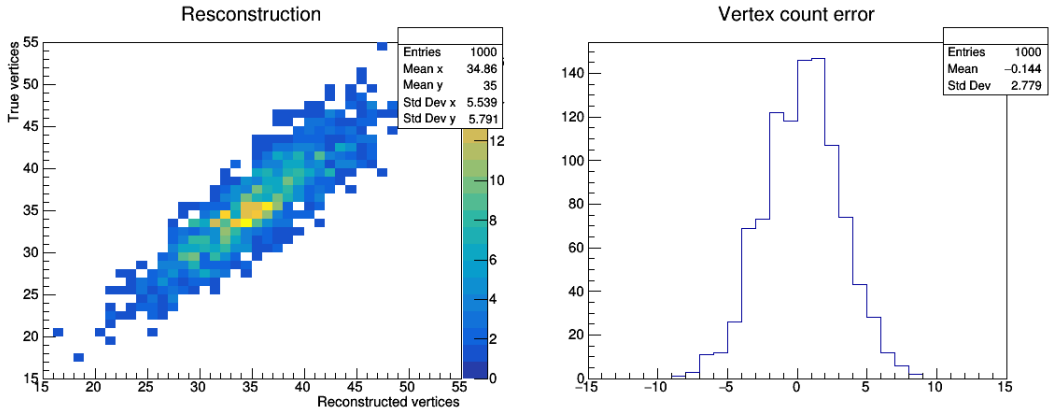

Figure 17: Reconstruction of vertices by the merge clustering algorithm (with merge distance set to 0.55mm, using the 'minimax' metric) $P_T^{min} = 2\text{GeV}$

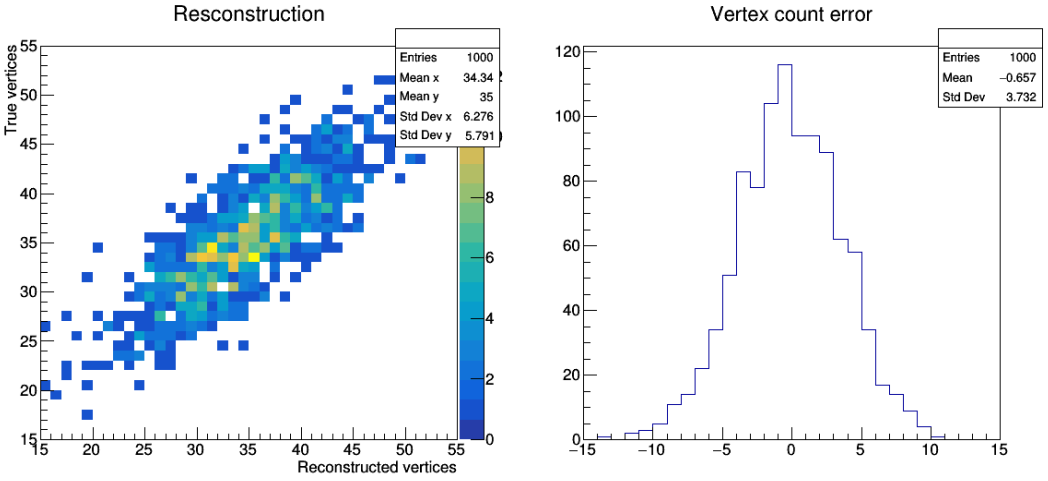

Figure 18: Reconstruction of vertices by the DBSCAN clustering algorithm (with $\epsilon = 0.1875\text{mm}$ and minPoints = 2) $P_T^{min} = 2\text{GeV}$

The first thing to note is the increase in the number of vertices; at the lower P_T^{min} more vertices remain after track reconstruction as expected. Also interesting to note is that the merge algorithm performs better than the DBSCAN algorithm at this P_T^{min}. This is perhaps due to the fact that the merge algorithm ensures that it is merging the most compatible tracks at each stage, and as such is density independent, whereas the DBSCAN algorithm traverses the set of tracks until it finds two areas that are distinctly separated by a low density region; we are perhaps seeing the high-density breakdown of this algorithm. The gap algorithm still performs a poor reconstruction. We plot the same again but for $P_T^{min} = 1\text{GeV}$ in figures 19, 20 and 21 below.

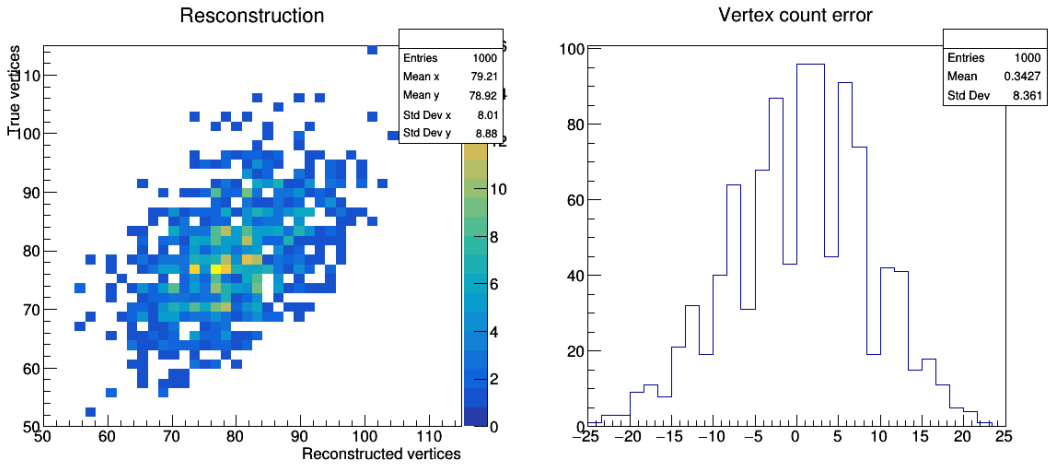

Figure 19: Reconstruction of vertices by the gap clustering algorithm (with gap set to 0.2812mm) $P_T^{min} = 1\text{GeV}$

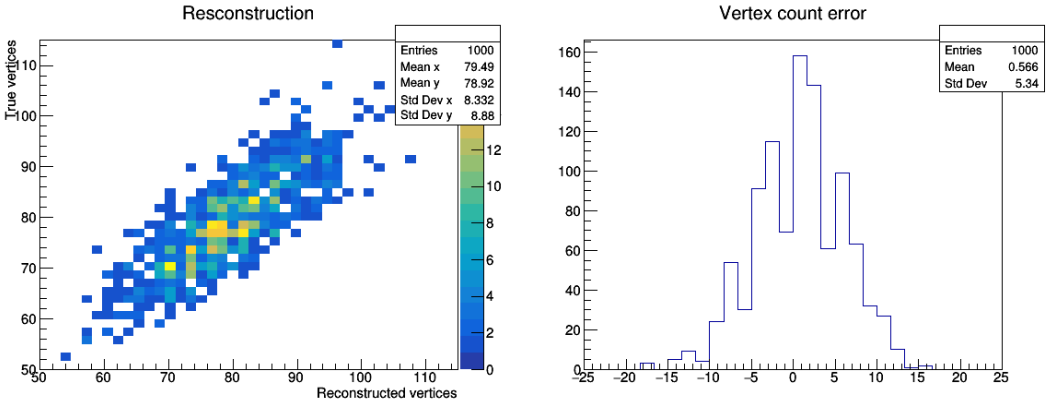

Figure 20: Reconstruction of vertices by the merge clustering algorithm (with merge distance set to 0.525mm, using the 'minimax' metric) $P_T^{min} = 1\text{GeV}$

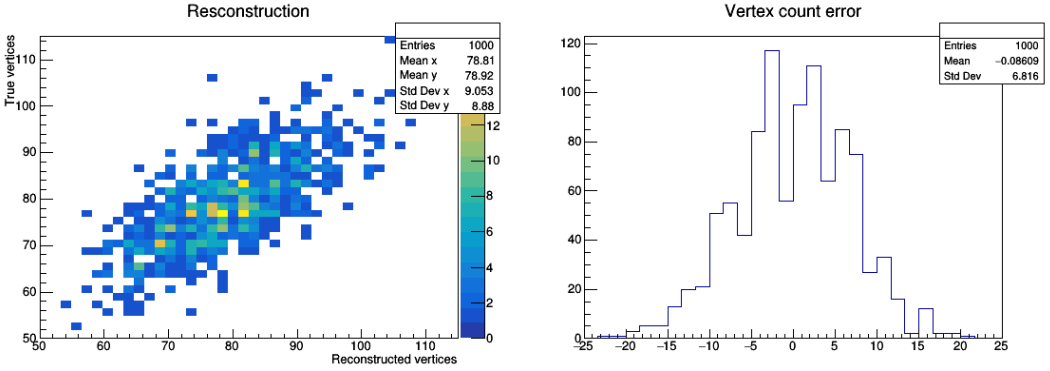

Figure 21: Reconstruction of vertices by the DBSCAN clustering algorithm (with $\epsilon = 0.16875$mm and minPoints = 2) $P_T^{min} = 1\text{GeV}$

We again see that the merge algorithm performs better than DBSCAN, and that the gap algorithm performs poorly. At lower values of P_T^{min} the merge algorithm also appears to have less sensitivity to the value of it's merge distance parameter, suggesting it is a better suited algorithm at these scales; the DBSCAN algorithm needs considerable fine tuning to perform well under these conditions.

5.1.2 Varying track smearing (σ)

As well as investigating how P_T^{min} affects the performance of these algorithms it is informative to look at how track resolution may affect their behaviour. To this end we show the same plots of reconstruction information at $P_T^{min} = 3$GeV, but for different track smearing. First we show these plots for $\sigma = 0.1$mm; this is shown in figures 22, 23 and 24 for the gap, merge and DBSCAN algorithms respectively.

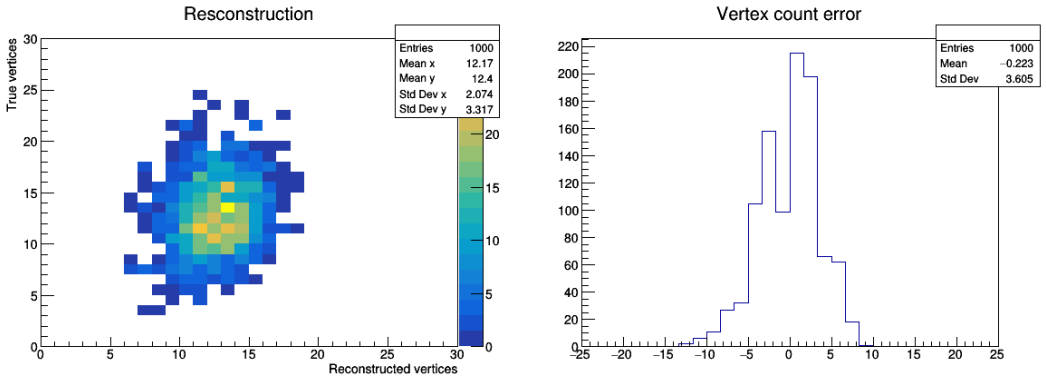

Figure 22: Reconstruction of vertices by the gap clustering algorithm (with gap set to 0.2812mm) $P_T^{min} = 3$GeV, $\sigma = 0.1$mm

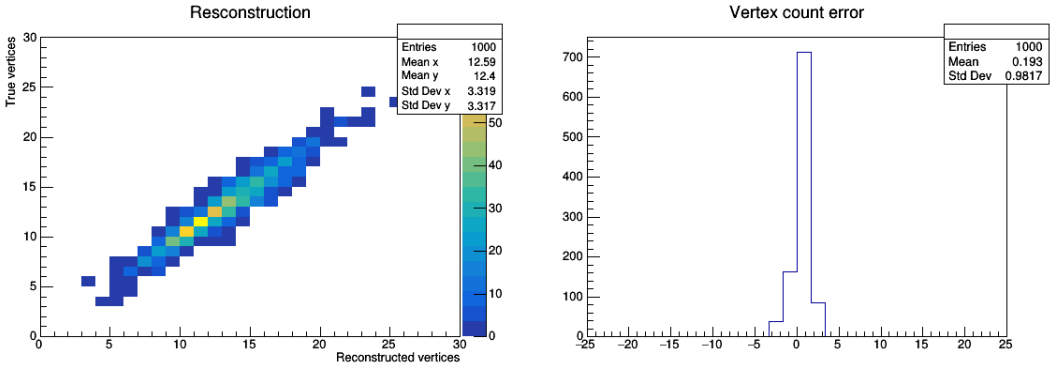

Figure 23: Reconstruction of vertices by the merge clustering algorithm (with merge distance set to 0.525mm, using the 'minimax' metric) $P_T^{min} = 3$GeV, $\sigma = 0.1$mm

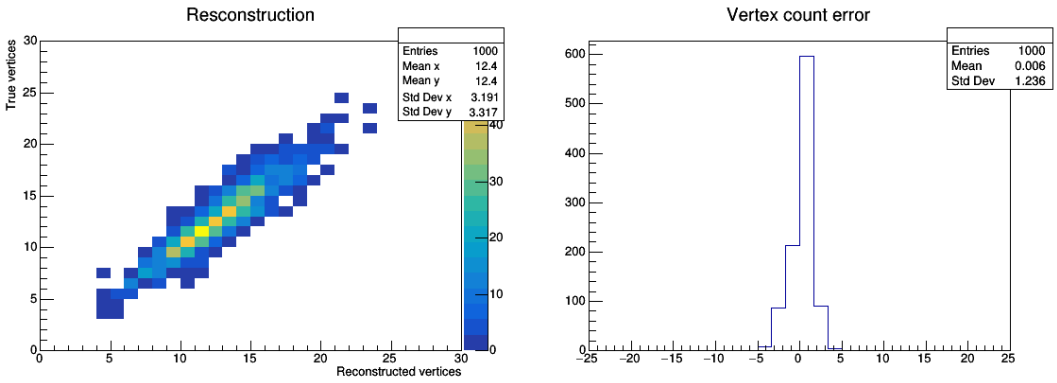

Figure 24: Reconstruction of vertices by the DBSCAN clustering algorithm (with $\epsilon = 0.16875$mm and minPoints = 2) $P_T^{min} = 3$GeV, $\sigma = 0.1$mm

The algorithms all perform better than they did at a smearing of 0.25mm (c.f section 4.1), which is to be expected. Interestingly however, the merge algorithm seems to perform better than the DBSCAN algorithm given this more accurate tracking information. This suggests, along with the improvement of the merge algorithm with a lower P_T^{min}, that the merge algorithm performs better with more complete information. However, the DBSCAN algorithm is better at reconstructing from less complete data than the merge algorithm in certain circumstances. To see if this trend continues we plot the same data for $\sigma = 1$mm in figures 25, 26 and 27.

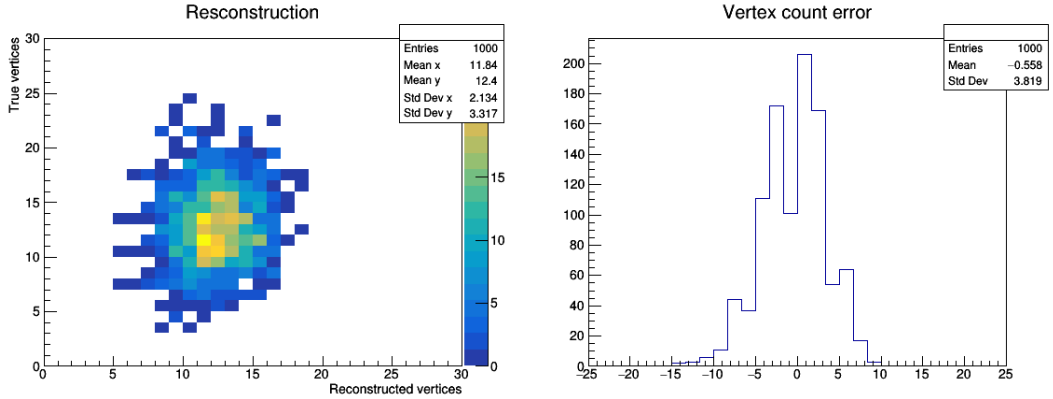

Figure 25: Reconstruction of vertices by the gap clustering algorithm (with gap set to 0.2812mm) $P_T^{min} = 3$GeV, $\sigma = 1$mm

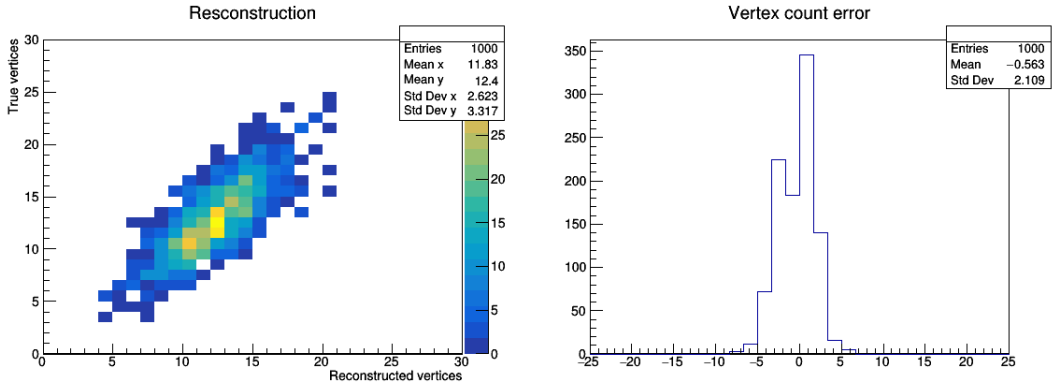

Figure 26: Reconstruction of vertices by the merge clustering algorithm (with merge distance set to 0.525mm, using the 'minimax' metric) $P_T^{min} = 3\text{GeV}$, $\sigma = 1\text{mm}$

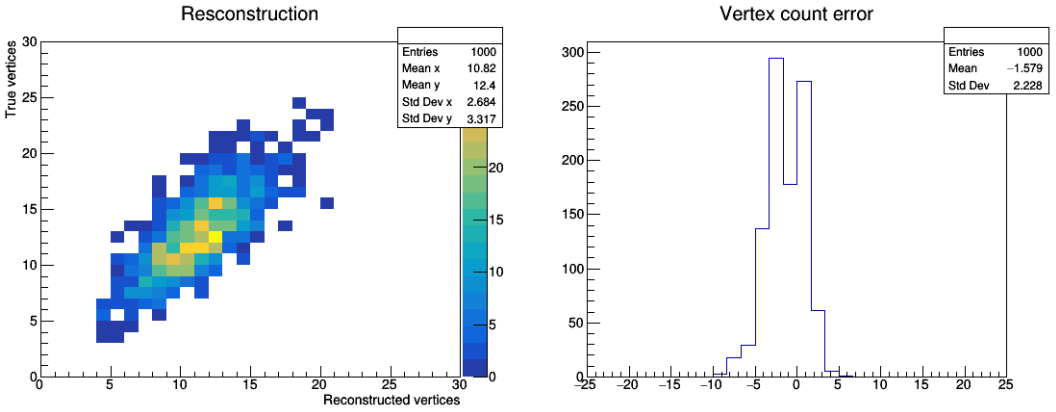

Figure 27: Reconstruction of vertices by the DBSCAN clustering algorithm (with $\epsilon = 0.16875\text{mm}$ and minPoints = 2) $P_T^{min} = 3\text{GeV}$, $\sigma = 1\text{mm}$

As expected, the algorithms perform considerably worse on highly smeared tracks. We see that the DBSCAN algorithm still performs slightly worse than the merge algorithm, as it did in the very low smearing limit. However in section 4.1, with $\sigma = 0.25\text{mm}$, the DBSCAN algorithm performed better than the merge algorithm. Interestingly, this suggests that the DBSCAN algorithm has a preferred operating range where it outperforms the merge algorithm; this range appears to be around $P_T^{min} \simeq 3\text{GeV}$ and $\sigma \in [0.25\text{mm}, 0.75\text{mm}]$. It is also interesting to see the effect that this increased smearing has on the position resolution of the vertex reconstruction; below we plot the same nearest reconstruction distance plots as in section 4.2 but for a track smearing of $\sigma = 1\text{mm}$ in figures 28, 29 and 30.

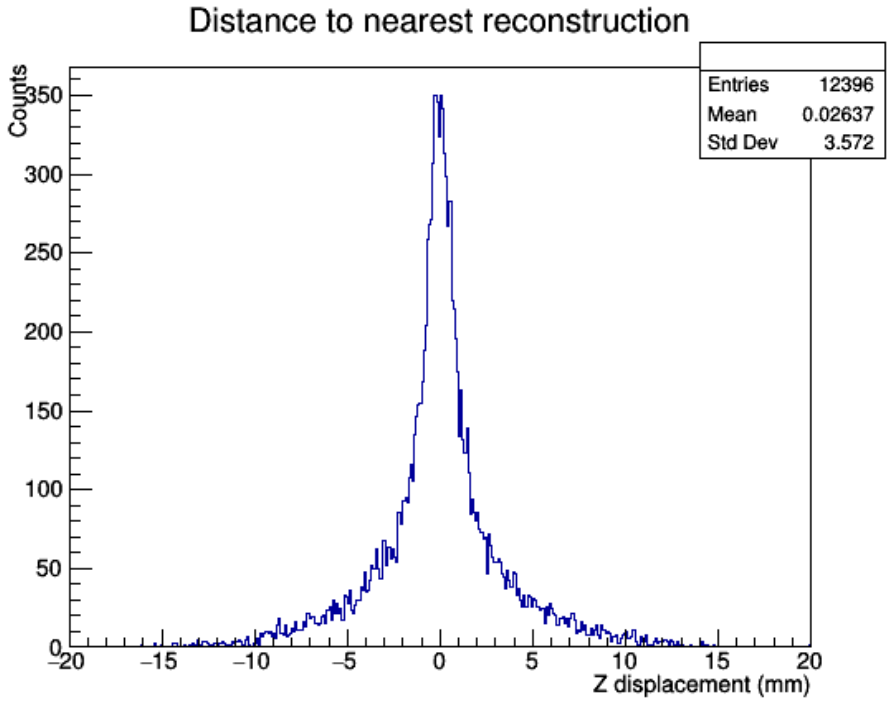

Figure 28: Distances to nearest reconstructed vertex, gap clustering algorithm (with gap set to 0.2812mm) $P_T^{min} = 3\text{GeV}$, $\sigma = 1\text{mm}$

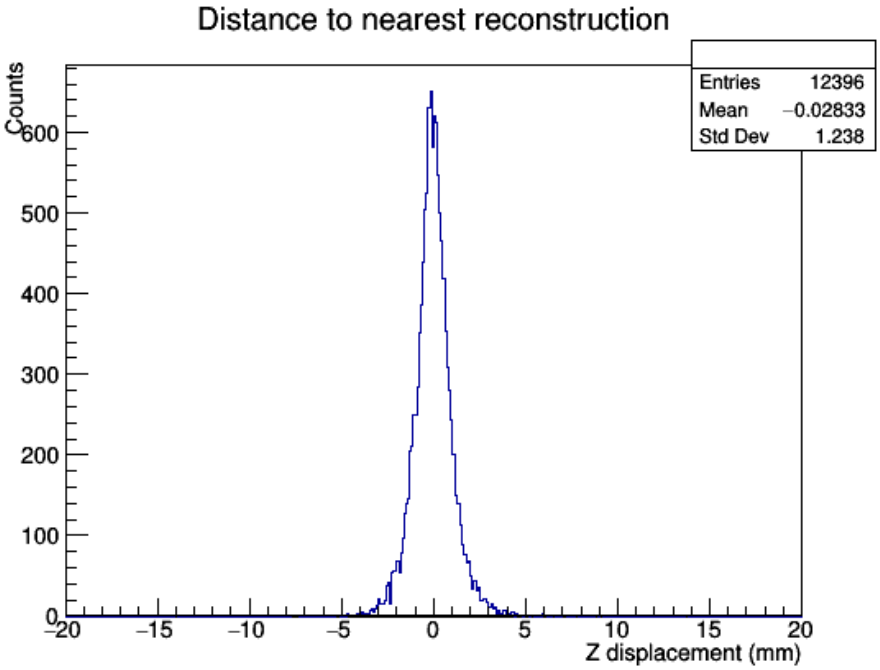

Figure 29: Distances to nearest reconstructed vertex, merge clustering algorithm (with merge distance set to 0.525mm, using the 'minimax' metric) $P_T^{min} = 3\text{GeV}$, $\sigma = 1\text{mm}$

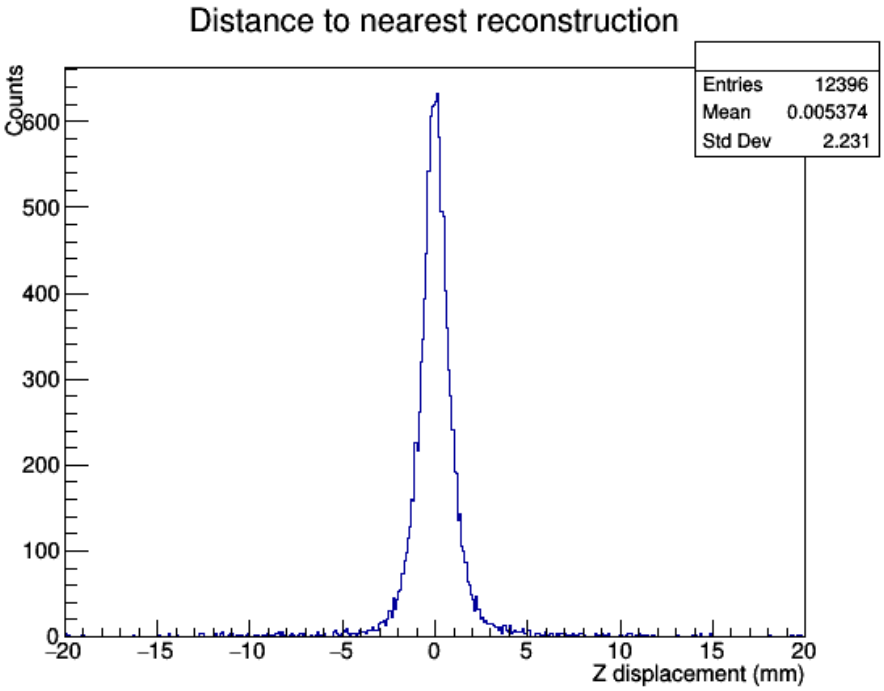

Figure 30: Distances to nearest reconstructed vertex, DBSCAN clustering algorithm (with $\epsilon = 0.16875$mm and minPoints = 2) $P_T^{min} = 3$GeV, $\sigma = 1$mm

We see that the increased smearing has propagated into an uncertainty in the reconstructed position of the vertices, which is to be expected. It is also interesting to note that the merge algorithm continues to out perform the DBSCAN algorithm in this high-smearing regime; this can be seen from comparing the standard deviation in plots 29 and 30. It seems that the merge algorithm is very robust against high displacements of reconstructed vertices from the truth; none are reconstructed with |displacement| > 5mm. This is not the case for the DBSCAN algorithm; a possible explanation of this is the so called 'chaining' effect, whereby the track density doesn't decrease enough to produce multiple vertices across a large chunk of the sample, resulting in vertices in this chunk being 'chained' together into one vertex. This may become apparent at high smearing because vertices will be smeared into one another, creating the possibility of large regions that exhibit small density variation.

5.2 Increased pileup

As well as track reconstruction effects it is interesting to see how the algorithms behave under increased pileup conditions, to test how robust they may be against changes in luminocity at the LHC. We again look at the correlation between the number of reconstructed vertices and true vertices at $P_T^{min} = 3$GeV and a track smearing of $\sigma = 0.25$mm, as in section 4.1, but we set the mean pileup to 200, 250 and 1000. We expect the algorithm performance at higher pileup to mimic that at a lower P_T cut, as this too increases the number of vertices that we deal with. Included is the full compliment of performance metrics for the most effective algorithm at each pileup; this includes the vertex count correlation and error plots, the vertex position error plots and the hard event track association percentage plots that we've seen in previous sections. The algorithm with the smallest standard deviation in vertex count errors ($\sigma_{vertex\ error}$) was

considered to be the most effective (this is the standard deviation of the bottom right graph in the below plots).

We first investigate a pileup of 200, at this pileup the merge algorithm again replaces the DBSCAN algorithm as the most effective (by a small margin of $\sigma_{vertex\ error} = 1.778$ for merge vs. 1.900 for DBSCAN); we are perhaps again seeing the high density breakdown of the DBSCAN algorithm. Figure 31 shows the performance metrics for the merge algorithm under these conditions.

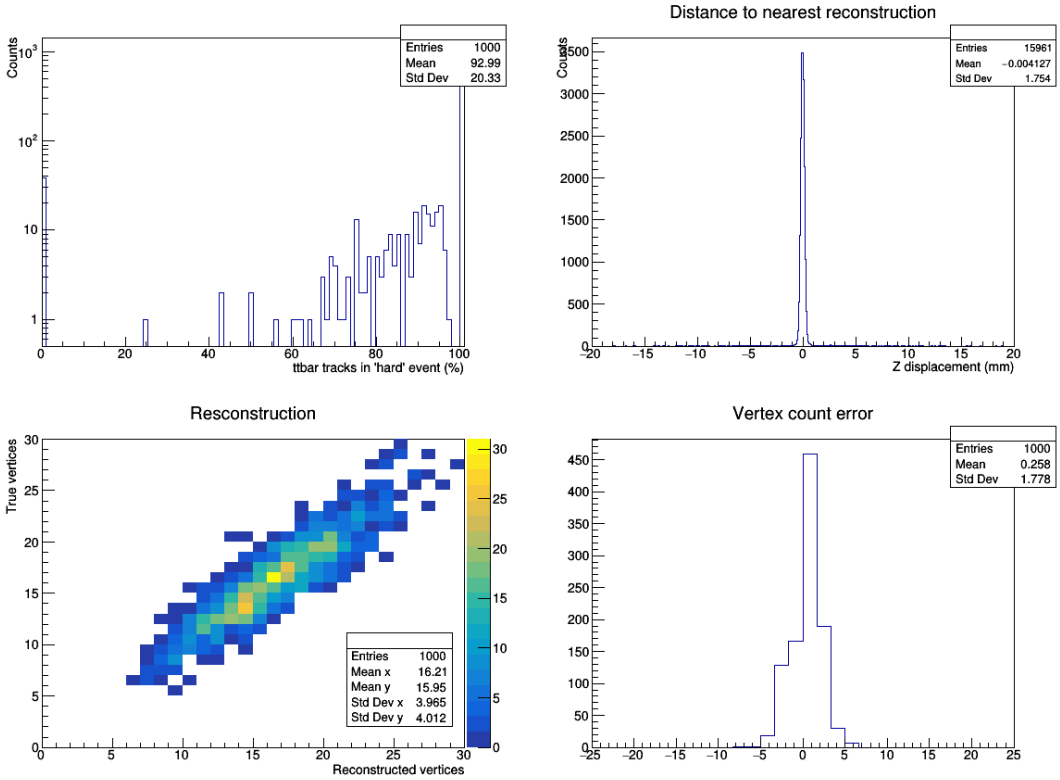

Figure 31: Performance metrics for the merge algorithm (with merge distance = 0.5mm) at 200 mean pileup, $P_T^{min} = 3$GeV, $\sigma = 0.25$mm

The first thing to note is the increase in number of reconstructable vertices, which is proportional to the increase in pilup as expected. The merge algorithm actually performs better under these higher pileup conditions than it did in section 4.1 with a pileup of 150. This is perhaps due to the less sporadic distribution of vertices that results from the increased number of vertices; the increased number of vertices effectively smooths out the vertex density across the sample (conditions under which the merge algotithm performs well, and the DBSCAN algorithm performs poorly). However, it can't quite match how the DBSCAN algorithm performed at lower piluep. We increase the pileup again to 250, at which point the results switch once again, and the DBSCAN algorithm performs better! The performance metrics for the DBSCAN algorithm at this pileup are shown in figure 32.

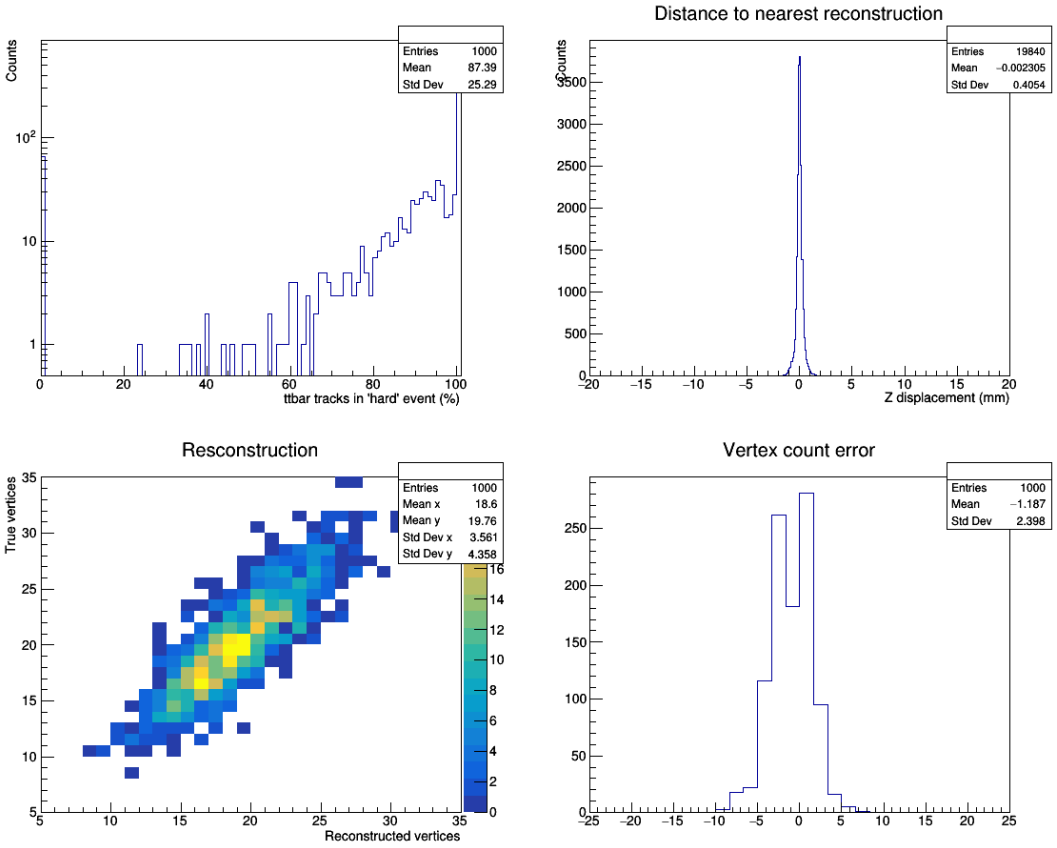

Figure 32: Performance metrics for the merge algorithm (with merge distance = 0.5mm) at 250 mean pileup, $P_T^{min} = 3\text{GeV}$, $\sigma = 0.25\text{mm}$

It seems that as we increase the piluep further still the merge algorithm hits a density limitation; one that has a stronger effect than that affecting the DBSCAN algorithm. It seems that which of these two algorithms will perform better under a given set of conditions is a difficult question to answer. Because both algorithms are now being somewhat density limited, the vertex count error has increased significantly ($\sigma_{vertex\ error}$ has increased from 1.778 to 2.398); although, if viewed as an error relative to the number of vertices, the effect is not so dramatic. Out of interest we investigate the algorithm performance under extremely high pileup conditions (1000 pileup), at this pileup the merge algorithm performs best; we plot it's performance characteristics in figure 33.

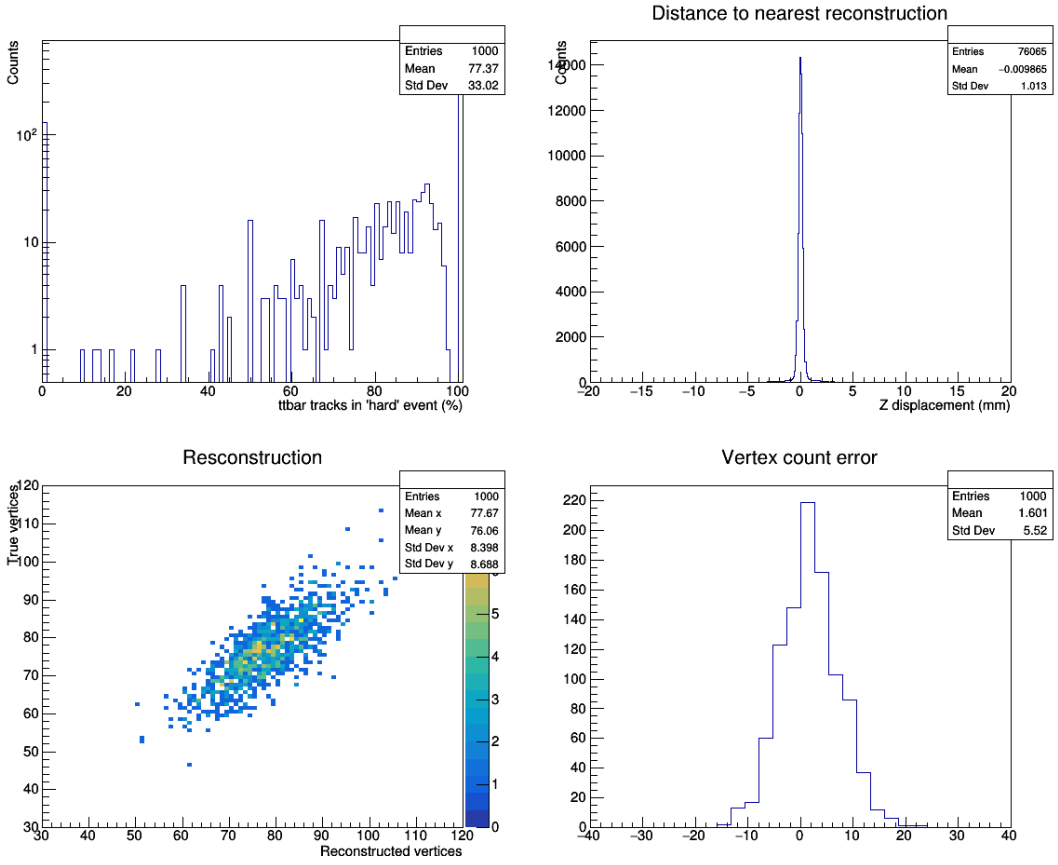

Figure 33: Performance metrics for the merge algorithm (with merge distance = 0.34mm) at 1000 mean pileup, $P_T^{min} = 3\text{GeV}$, $\sigma = 0.25\text{mm}$

At this level of pileup it is encouraging to see that the algorithm is still able to give us some correlation between the truth and it's reconstruction. It is interesting to note that, despite considerable fine tuning, it was impossible to get the DBSCAN algorithm to reconstruct the correct number of vertices, it would always under-reconstruct, although it still showed a decent correlation. In contrast, the merge algorithm was relatively easy to provide with parameters that would produce reasonable results.

6 Applications of vertex information

6.1 Pileup energy reconstruction

We now turn our attention to what kind of results can be obtained once we have this vertex reconstruction information. One of the first pieces of information that could be obtained is an estimate on the amount of pileup energy that goes into the detector; an example of where this is useful information is in pileup energy subtraction for jet reconstruction. To calculate the pileup energy we simply sum up the energy of all of the particles that make it to the outer edge of the detector. We apply a P_T cut of 100MeV to obtain these particles from the list of all particles, as particles with $P_T < 100$MeV will be wound up in the magnetic field and not make it out to the detector; we also apply an η cut of 5 to model the loss of particles into the very forward region of the detector (along the beampipe). We then look at the correlation of this energy to the number of pileup vertices; this is shown in figure 34.

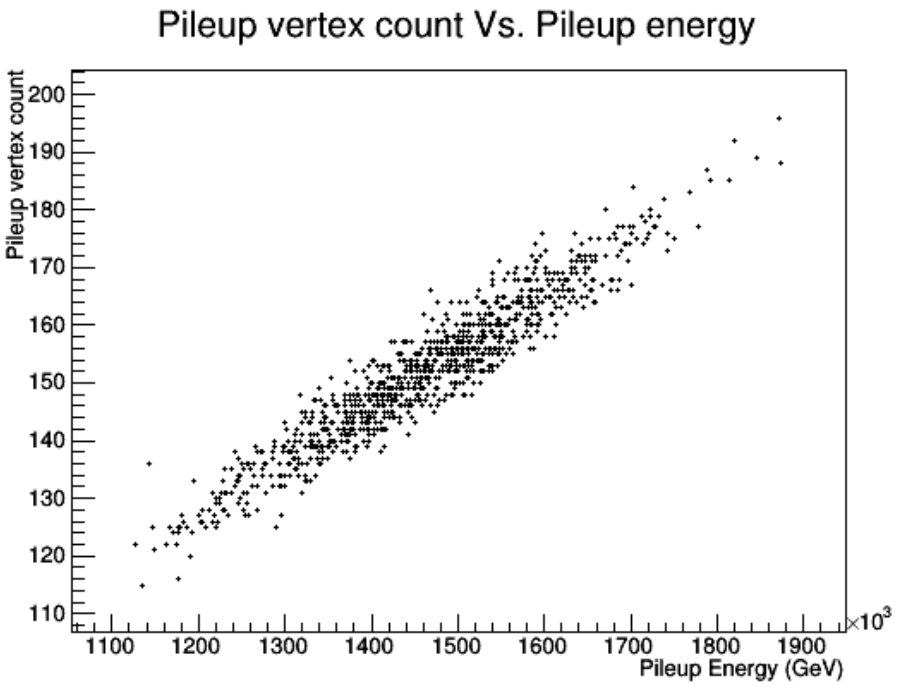

Figure 34: Pileup energy vs. number of pileup vertices (1000 events, mean pileup of 150)

We see that this shows a strong correlation, as we might expect (It is not a perfect correlation as a result of the cuts that we are applying to the particles that are used to make up this plot). However, this is not the information that we will have access to, as we must apply a P_T cut to the tracks first. To see how well we might construct the pileup energy from the number of vertices that still exist after the P_T cut, we plot the correlation of pileup energy to vertices that have at least two tracks with $P_T > P_T^{min} =$ 3GeV in figure 35. This correlation tells us how well we could reconstruct the pileup energy if our vertex reconstruction algorithm performed perfectly.

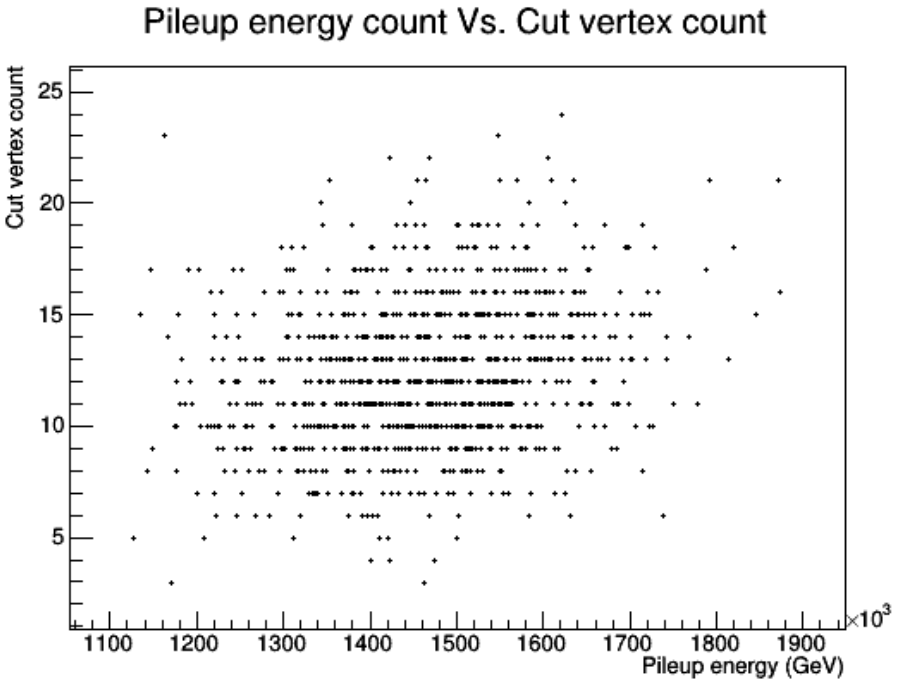

Figure 35: Pileup energy vs. number of pileup vertices (1000 events, mean pileup of 150) vertices cut if they don't have at least two tracks with $P_T > 3$Gev

We see that any correlation is lost with the P_T cut. This means that any attempt to reconstruct pileup energy from reconstructed vertex count will be futile at a P_T cut of 3GeV; however, the question still remains if this is the case at lower P_T cuts. We plot the same information as figure 35 but for P_T cuts of 2 and 1GeV in figures 36 and 37 respectively.

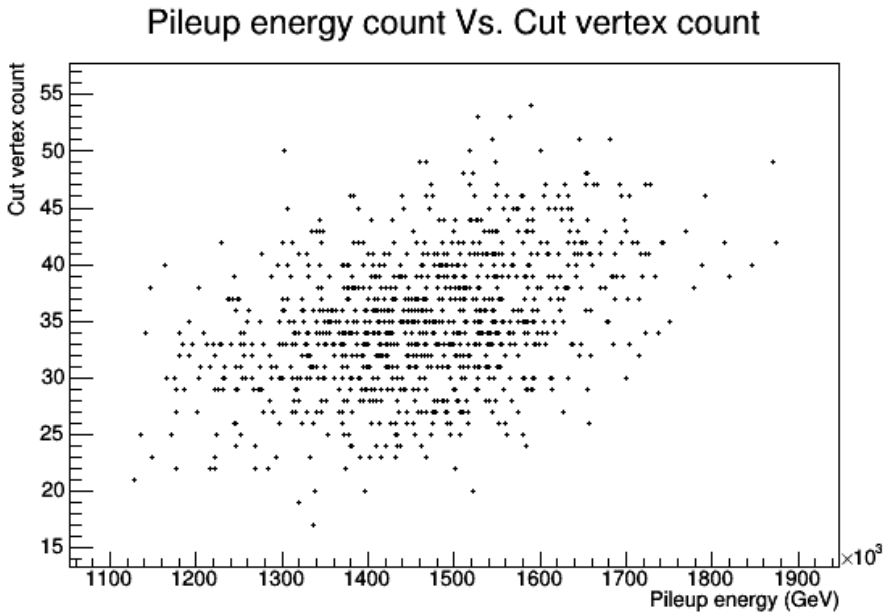

Figure 36: Pileup energy vs. number of pileup vertices (1000 events, mean pileup of 150) vertices cut if they don't have at least two tracks with $P_T > 2$Gev

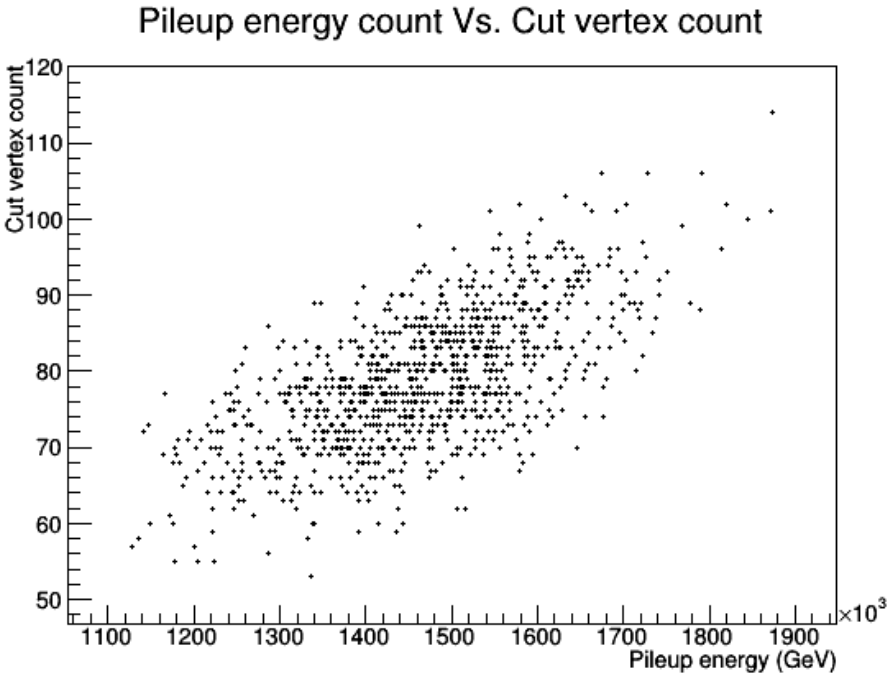

Figure 37: Pileup energy vs. number of pileup vertices (1000 events, mean pileup of 150) vertices cut if they don't have at least two tracks with $P_T > 1$Gev

We see that the correlation improves with lower P_T cut as expected. However, it is still not correlated enough to be usable as a good estimator of pileup energy. It seems

that getting to the pileup energy via the number of vertices is an unrealistic goal at any significant P_T cut. There may be other ways to get to pileup energy from tracking information, however we do not investigate this further in this study.

7 Conclusion

We have investigated the performance and implementation of three vertex reconstruction algorithms; the gap, merge and DBSCAN algorithms. Close attention was paid to how well these algorithms perform under the best estimate of operating conditions at the HL-LHC (These conditions correspond to a pileup of 150 13TeV minimum bias events, in a gaussian distribution with $\sigma = 50$mm along the beamline). The online tracking system response was modelled, including a P_T^{min} of 3GeV (implemented as a P_T cut on tracks) and a track position smearing of $\sigma = 0.25$mm. It was found that the P_T cut in the tracking system greatly reduces the information available to use for vertex reconstruction, as we loose around three orders of magintude in the number of tracks when a P_T cut of 3GeV is applied (see figures 4 and 5). This means that only around 8% of pileup interaction vertices leave a signature in the tracking system. The ability of the algorithms to reconstruct the remaining interaction vertices has been tested in various ways; they reconstruct the number of interaction vertices, their positions and the tracks that are associated to them reasonably well. The number of these remaining 'reconstructable' vertices appears to be the hardest quantity to reconstruct, with the best algorithm under HL-LHC conditions (the DBSCAN algorithm) being able to reconstruct this number with a standard deviation of 1.384 vertices (out of a mean of 12.4 reconstructable vertices). We considered the performance of the algorthim to be well characterised by the error in the reconstruction of the number of 'reconstructable' vertices. The position of the vertices seems to be an easier quantity to reconstruct, with the DBSCAN algorithm being able to reconstruct vertex position to within a millimetre most of the time (a standard deviation of 0.9211 mm is achieved in reconstruction error). The tracks associated to a $t\bar{t}$ production vertex were compared to the truth, and it was found that most of the time it was possible to correctly associate over 90% of the $t\bar{t}$ tracks. The effects of varying the input sample were investigated, with either the DBSCAN or merge algorithm performing best depending on the sample, it being largely impossible to predict which would perform better. The input sample was varied to represent better, and worse, track reconstructions by varying P_T^{min} for track reconstruction and the position smearing, σ, of the resulting tracks. The impact of this on the vertex reconstruction was investigated; the algorithms typically performed worse when provided with lower resolution tracking information (higher σ), but the reconstruction accuracy of 'reconstructable' vertices seemed to be largely unaffected by varying tracking efficiencies (varying P_T^{min}), as well as by increased pileup density; indeed, the merge algorithm was shown to continue to work reasonably even with 1000 pileup events. The possibility of using the reconstructed vertex information to reconstruct pileup energy was investigated, but was found to be very unlikely to produce useful results.

8 References

[1] *New vertex reconstruction algorithms for CMS.*
R. Frühwirth, W. Waltenberger, K. Prokofiev, T. Speer, P. Vanlaer, E. Chabanat and N. Estre,
http://www.slac.stanford.edu/econf/C0303241/proc/papers/TULT013.PDF

[2] *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.*
Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu,
http://www2.cs.uh.edu/ ceick/7363/Papers/dbscan.pdf

[3] *Track and Vertex Reconstruction: From Classical to Adaptive Methods.*
Are Strandlie, Rudolf Frühwirth
http://www.hephy.at/fileadmin/user_upload/Fachbereiche/ASE/Strandlie.pdf

9 Appendix

9.1 PYTHIA $t\bar{t}$ card file

```
!Card file for ttbar generation

Main:numberOfEvents = 5000
Beams:eCM = 13000
Next:numberCount = 100
Random:setSeed = on
Random:setSeed = 10

!Turn on processes that we're interested in
Top:gg2ttbar = on      ! g g -> t tbar
```

9.2 PYTHIA minimum bias card file

```
!Card file for pileup generation

! 1) Settings that will be used in a main program.
Main:numberOfEvents = 10000        ! number of events to generate
Main:timesAllowErrors = 3          ! abort run after this many flawed events

! 2) Settings related to output in init(), next() and stat().
Init:showChangedSettings = on      ! list changed settings
Init:showAllSettings = off         ! list all settings
Init:showChangedParticleData = on  ! list changed particle data
Init:showAllParticleData = off     ! list all particle data
Next:numberCount = 100             ! print message every n events
Next:numberShowLHA = 1             ! print LHA information n times
Next:numberShowInfo = 1            ! print event information n times
Next:numberShowProcess = 1         ! print process record n times
Next:numberShowEvent = 1           ! print event record n times
Stat:showPartonLevel = on          ! additional statistics on MPI

! 3) Beam parameter settings. Values below agree with default ones.
Beams:idA = 2212                   ! first beam, p = 2212
Beams:idB = 2212                   ! second beam, p = 2212
Beams:eCM = 13000                  ! CM energy of collision

! 4a) Generate soft QCD processes.
SoftQCD:all = on                   ! Allow total sigma = elastic/SD/DD/ND

! 4b) Other settings.
Tune:pp = 5                         ! use Tune 5
```

9.3 Delphes Card File

```
########################################
# Order of execution of various modules
########################################

set ExecutionPath {
  PileUpMerger
  ParticlePropagator
  ChargedHadronTrackingEfficiency
  ElectronTrackingEfficiency
  MuonTrackingEfficiency
  ChargedHadronMomentumSmearing
  ElectronMomentumSmearing
  MuonMomentumSmearing
  TrackMerger
  TreeWriter
}

###############
# PileUp Merger
###############

module PileUpMerger PileUpMerger {
  set InputArray Delphes/stableParticles

  set ParticleOutputArray stableParticles
  set VertexOutputArray vertices

  # pre-generated minbias input file
  set PileUpFile MinBias.pileup

  # average expected pile up
  set MeanPileUp 150

   # maximum spread in the beam direction in m
  set ZVertexSpread 0.25

  # maximum spread in time in s 800E-12
  set TVertexSpread 1E-15

  # pileup distribution formula f(z,t) (z,t need to be respectively given in m,s)
  # Guassian smearing with sigma = 0.05
  set VertexDistributionFormula {exp(-(z^2/(2*0.05^2)))}

}

#################################
# Propagate particles in cylinder
#################################

module ParticlePropagator ParticlePropagator {
  set InputArray PileUpMerger/stableParticles

  set OutputArray stableParticles
  set ChargedHadronOutputArray chargedHadrons
  set ElectronOutputArray electrons
  set MuonOutputArray muons

  # radius of the magnetic field coverage, in m
  set Radius 1.29
  # half-length of the magnetic field coverage, in m
  set HalfLength 3.00

  # magnetic field
  set Bz 3.8
}
```

```
####################################
# Charged hadron tracking efficiency
####################################

module Efficiency ChargedHadronTrackingEfficiency {
  set InputArray ParticlePropagator/chargedHadrons
  set OutputArray chargedHadrons

  # add EfficiencyFormula {efficiency formula as a function of eta and pt}
  # tracking efficiency formula for charged hadrons is set to 1, we model this in later analysis
  set EfficiencyFormula {1}
}

##############################
# Electron tracking efficiency
##############################

module Efficiency ElectronTrackingEfficiency {
  set InputArray ParticlePropagator/electrons
  set OutputArray electrons

  # set EfficiencyFormula {efficiency formula as a function of eta and pt}
  # tracking efficiency formula for electrons is set to 1, we model this in later analysis
  set EfficiencyFormula {1}
}

##########################
# Muon tracking efficiency
##########################

module Efficiency MuonTrackingEfficiency {
  set InputArray ParticlePropagator/muons
  set OutputArray muons

  # set EfficiencyFormula {efficiency formula as a function of eta and pt}
  # tracking efficiency formula for muons set to 1, we model this in later analysis
  set EfficiencyFormula {1}
}

########################################
# Momentum resolution for charged tracks
########################################

module MomentumSmearing ChargedHadronMomentumSmearing {
  set InputArray ChargedHadronTrackingEfficiency/chargedHadrons
  set OutputArray chargedHadrons

  # set ResolutionFormula {resolution formula as a function of eta and pt}
  # resolution formula for charged hadrons
  # based on arXiv:1405.6569
  set ResolutionFormula {                  (abs(eta) <= 0.5) * (pt > 0.1) * sqrt(0.01^2 + pt^2*1.5e-4^2) +
                         (abs(eta) > 0.5 && abs(eta) <= 1.5) * (pt > 0.1) * sqrt(0.015^2 + pt^2*2.5e-4^2) +
                         (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1) * sqrt(0.025^2 + pt^2*5.5e-4^2)}
}

####################################
# Momentum resolution for electrons
####################################

module MomentumSmearing ElectronMomentumSmearing {
  set InputArray ElectronTrackingEfficiency/electrons
  set OutputArray electrons

  # set ResolutionFormula {resolution formula as a function of eta and energy}
  # resolution formula for electrons
  # based on arXiv:1405.6569
  set ResolutionFormula {                  (abs(eta) <= 0.5) * (pt > 0.1) * sqrt(0.03^2 + pt^2*1.3e-3^2) +
                         (abs(eta) > 0.5 && abs(eta) <= 1.5) * (pt > 0.1) * sqrt(0.05^2 + pt^2*1.7e-3^2) +
                         (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1) * sqrt(0.15^2 + pt^2*3.1e-3^2)}
```

```
}

################################
# Momentum resolution for muons
################################

module MomentumSmearing MuonMomentumSmearing {
  set InputArray MuonTrackingEfficiency/muons
  set OutputArray muons

  # set ResolutionFormula {resolution formula as a function of eta and pt}
  # resolution formula for muons
  set ResolutionFormula {                  (abs(eta) <= 0.5) * (pt > 0.1) * sqrt(0.01^2 + pt^2*1.0e-4^2) +
                         (abs(eta) > 0.5 && abs(eta) <= 1.5) * (pt > 0.1) * sqrt(0.015^2 + pt^2*1.5e-4^2) +
                         (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1) * sqrt(0.025^2 + pt^2*3.5e-4^2)}
}

###############
# Track merger
###############

module Merger TrackMerger {
# add InputArray InputArray
  add InputArray ChargedHadronMomentumSmearing/chargedHadrons
  add InputArray ElectronMomentumSmearing/electrons
  add InputArray MuonMomentumSmearing/muons
  set OutputArray tracks
}

##################
# ROOT tree writer
##################

module TreeWriter TreeWriter {
  add Branch Delphes/allParticles Particle GenParticle
  add Branch TrackMerger/tracks Track Track
  add Branch PileUpMerger/vertices Vertex Vertex
  add Branch PileUpMerger/stableParticles StableParticles GenParticle
}
```

Pileup event

run_num: 0
lumi_section: 0
event_num: 0

9.4 Minimum bias decay diagram