Framing secondary vertexing as a clustering problem: First results with Point Cloud and HyperGraph

Diptaparna Biswas, Markus Cristinziani, Vadim Kostyukhin

Universität Siegen

HEP Experimental High Energy Physics

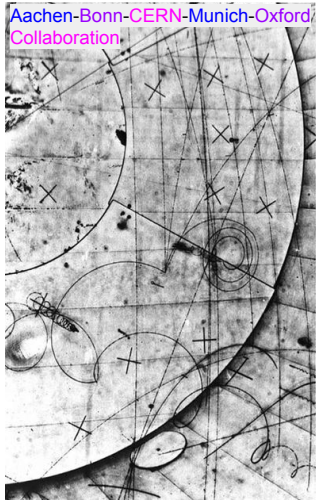
CPPS Center for Particle Physics Siegen

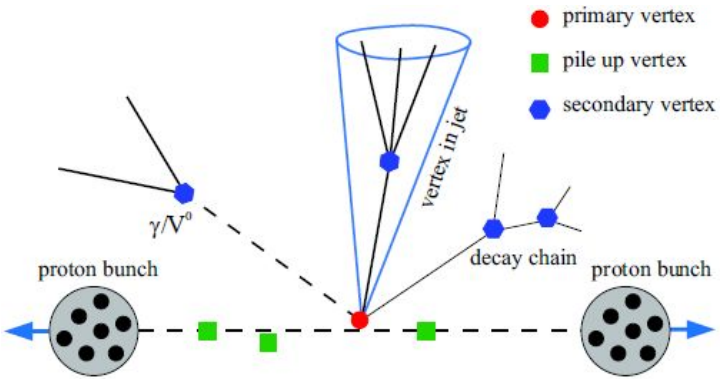
STAR Sensing and Sensibility
Transcending Disciplines for a Responsible Future

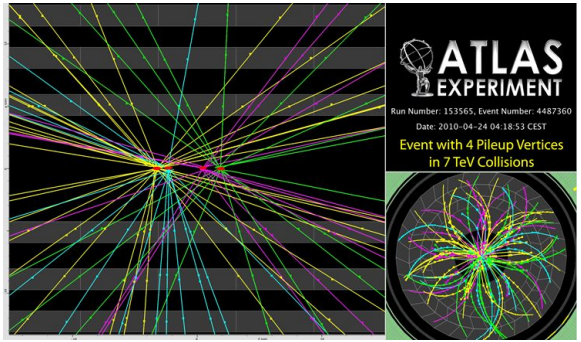
TrackOpt Collaboration Meeting
Ilmenau
11th March 2026

Vertexing and the HL–LHC

- Vertexing: Reconstructing particle interaction or decay points.
  - ➢ Crucial for identifying primary and secondary vertices.
  - ➢ Essential for flavor tagging (b/c quarks) and LLP searches.
- HL–LHC: Significant luminosity increase for precision studies.
  - ➢ Increased luminosity = increased challenges for vertexing.
  - ➢ Dramatic rise in pile-up (140-200 collisions/bunch crossing).



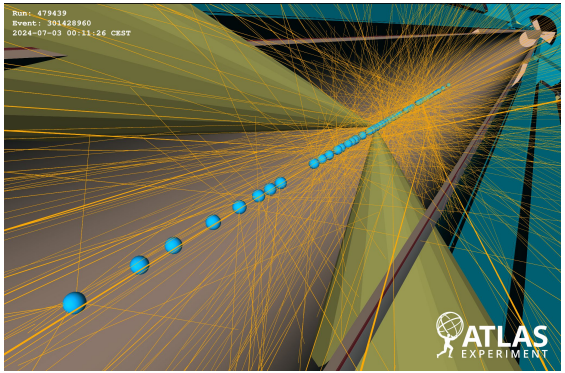


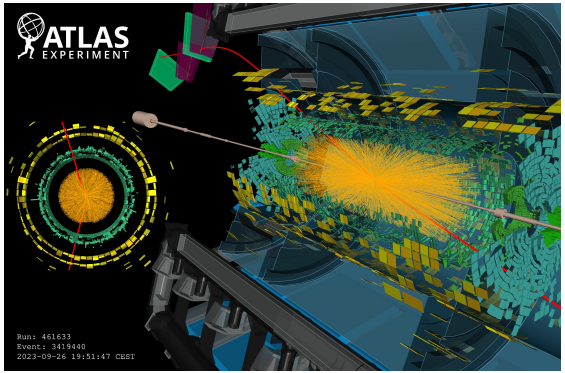


Vertexing in high luminosity environment

- The primary challenge: Pile-Up
  - ➢ Increased vertex density along the beamline.
  - ➢ Distinguishing hard scatter vertex from pile-up.

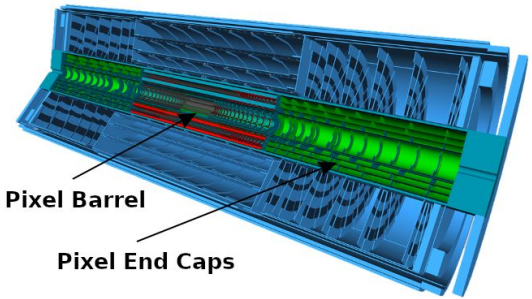
  } PV
  - ➢ Track-vertex association becomes highly complex.
  - ➢ Merged and split vertex errors.
- Impact of Increased Track Density
  - ➢ Makes any kind of pattern recognition challenging.
  - ➢ Increased combinatorial complexity of vertex finding.
  - ➢ Potential for ghost tracks and fake vertices.



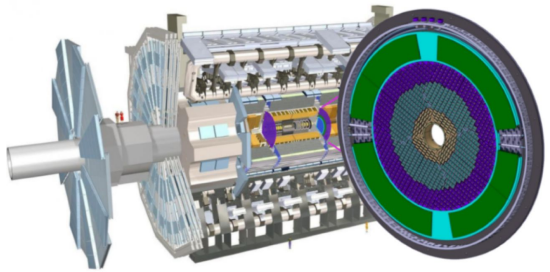


Maintaining and enhancing vertexing precision

- Importance of impact parameter and spatial resolution.
  - ➢ ATLAS Inner Tracker (ITk): increased granularity.
- Leveraging time information (4D-vertexing).
  - ➢ ATLAS HGTD and CMS MTD: precise timing information.



ATLAS ITk



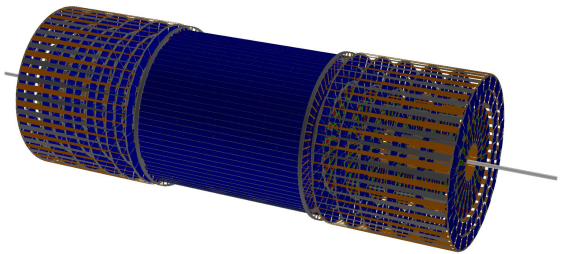
ATLAS HGTD

ACTS: *A Common Tracking Software*

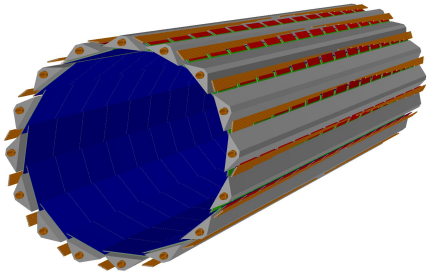
- Evolved from *ATLAS Common Tracking Software*
- Key Features:
  - ➢ Flexible tracking geometry description.
  - ➢ Simple and efficient event data model.
  - ➢ Algorithms for seed finding, track propagation, track fitting and primary vertexing.
- Designed in modern C++ (C++17/20): emphasis on parallel execution.
- Existing primary vertexing algorithms in ACTS:
  - ➢ Iterative Vertex Finder (IVF)
  - ➢ Adaptive Multi-Vertex Finder (AMVF)

ODD: Open Data Detector

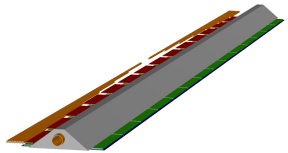
- Designed for algorithm research and development.
  - ➢ Based on the detector used in the TrackML challenge.
  - ➢ Provides a template for an HL-LHC detector, inspired by ATLAS Inner Tracker (ITk) upgrade.
- ODD serves as a crucial platform for testing and benchmarking ACTS track reconstruction algorithms. It provides a full reconstruction chain.
  - ➢ We extend the ACTS+ODD ecosystem for our secondary vertexing research for HL-LHC.



The full Open Data Detector



Innermost layer (r = 36 mm)



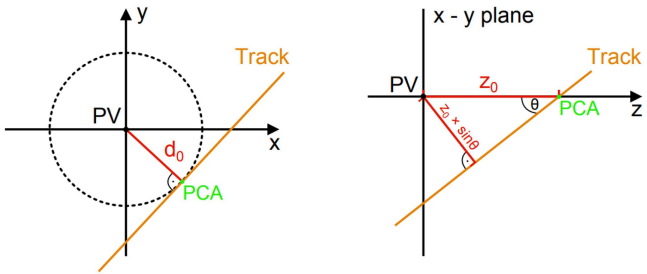
A single stave

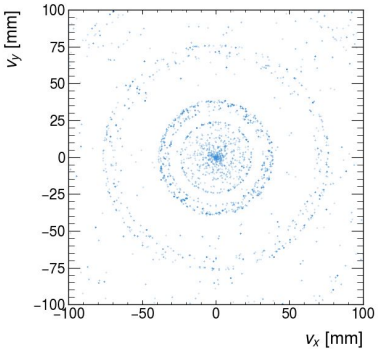
Preparation of synthetic dataset

- Inside the ACTS toolkit, the $pp \rightarrow t\bar{t}$ process is generated using Pythia 8.
- Full Geant4 simulation of the OpenDataDetector geometry.
- The value of **pile-up** is chosen to be **200**.
- Built-in track and vertex reconstruction is performed after simulation.
- Each event takes around ~ 5 minutes, for gen+sim+reco.

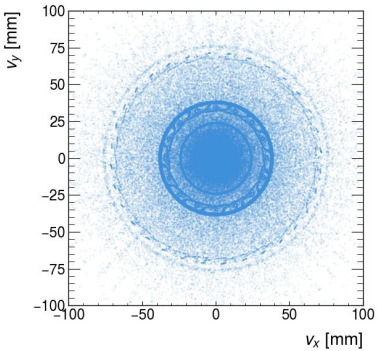
Hadronic interactions in ODD

- Secondary vertex finding should work in the presence of material interaction between hadrons and detector layers.
- To enhance the number of tracks originating from such interactions, the ODD sim+reco chain has been modified:
  - ➢ `SeedFinderConfig`
    - d_0 : 5 mm→20 mm
  - ➢ `ParticleSelectorConfig`
    - ρ : 24 mm→1000 mm





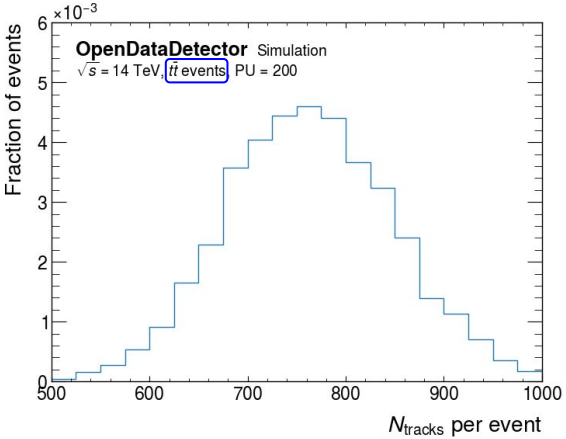
From simulated particles

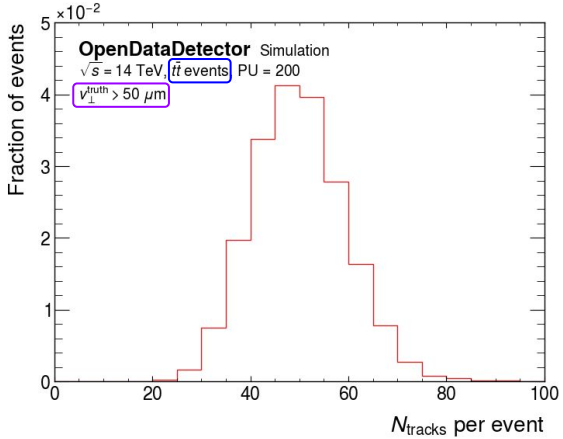


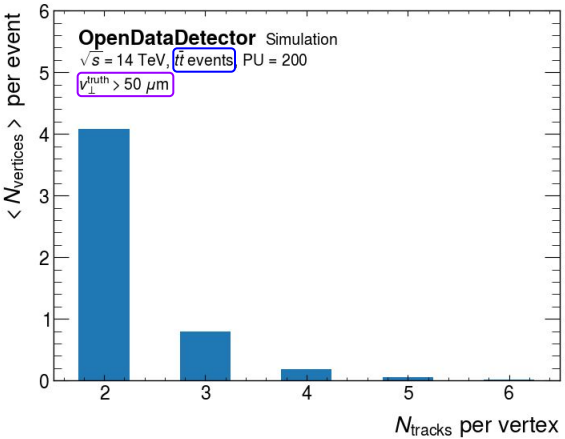
From reconstructed particles

ρ refers to the maximum radial distance in the cylindrical coordinate system.

Understanding the ODD dataset





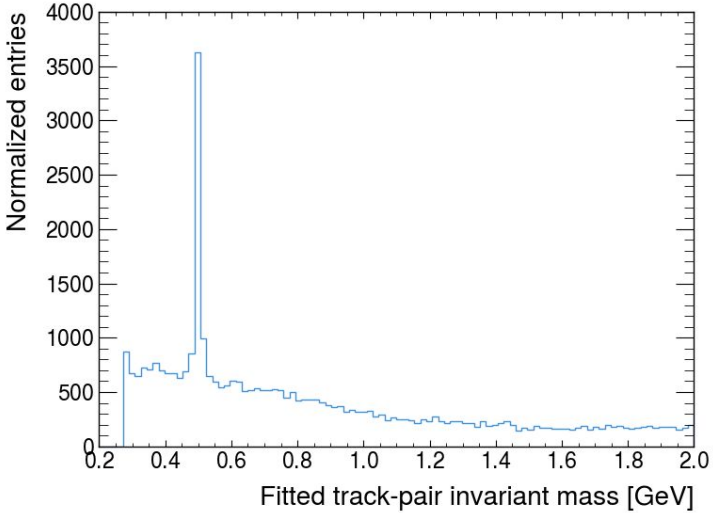


- Average number per event:
  - ➢ Total tracks: ~750
    - For the training dataset, only events with ≤ 1000 tracks have been selected.
  - ➢ Non-prompt tracks: ~50
  - ➢ ~4 two-track vertices and 1-2 multi-track vertices.

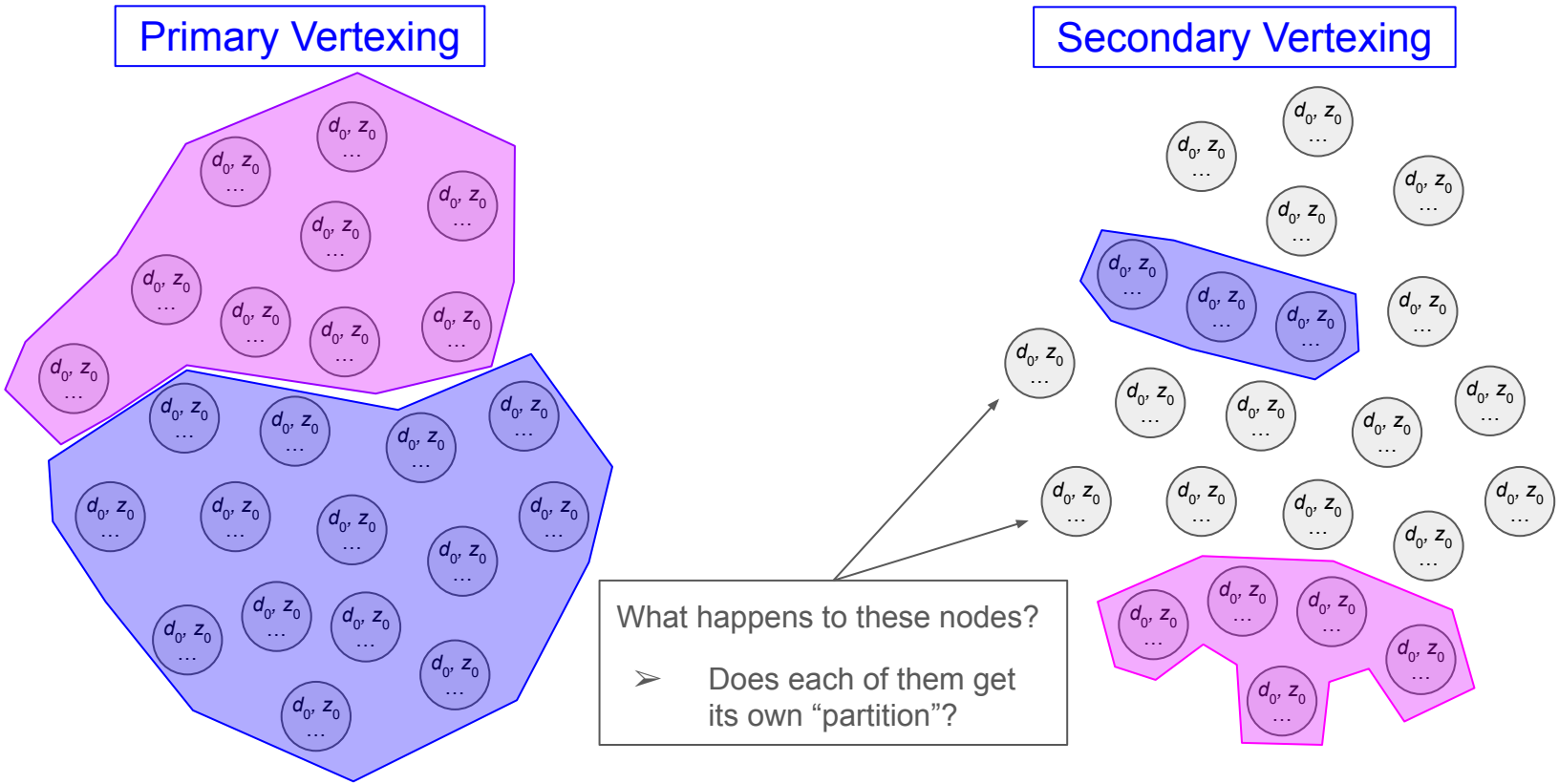
Calculating track-pair compatibility matrix

Billoir fit can be performed between each pair of tracks:

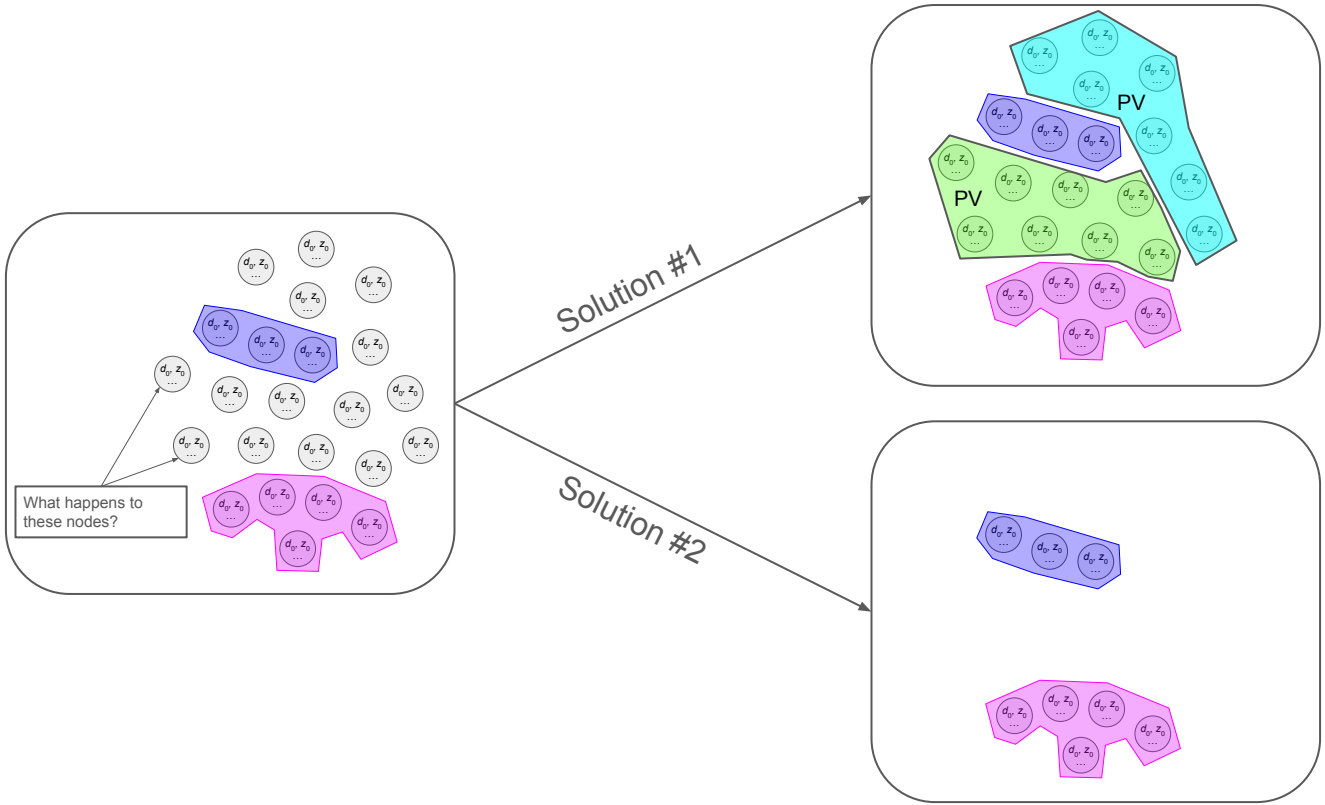
- 5 parameters from each fitted track, along with their (co)-variances, are used: d_0, z_0, ϕ, θ, q/p.
- The primary vertex has been chosen at (0,0,0), with a very large uncertainty in its z-coordinate.
- The fit returns the vertex position, mass, and χ^2 as a measure of the goodness of fit.



Framing Vertex finding as a graph partitioning problem



SV finding is not really a graph partitioning problem



Perform PV and SV finding simultaneously

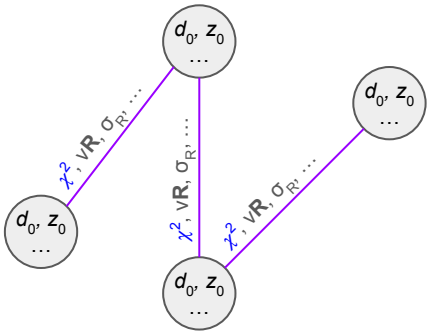
Get rid of the isolated nodes, *possibly in a soft way*

Framing vertex finding as a clustering problem

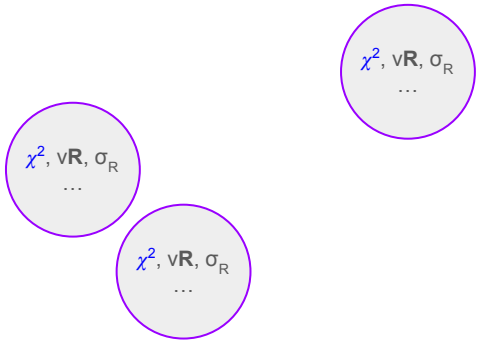
- Most obvious approach: A vertex is a cluster of **tracks**.
  - ➢ Consider the tracks as nodes.
    - Track params are node features.
    - Different quantities from track-pair Billoir fit can be taken as edge features.
  - ➢ This is the chosen strategy for MaskFormer as well as the previous primary-vertexing work.
- Alternate approach: A vertex is a cluster of one or more **two-track vertices**.
  - ➢ Each track-pair satisfying **Billoir fit** is a node (in a Point Cloud).
    - Different quantities from track-pair Billoir fit are node features.
    - No edge feature. The Euclidean distance might be taken as one.
  - ➢ Immediately offers us a bounding box heuristic.
  - ➢ For two-track vertices, the performance is guaranteed to be at least as good as Billoir fit.
  - ➢ Trivial to implement using `sklearn.cluster.DBSCAN` to establish a baseline.
    - Useable with better clustering techniques (e.g. lifted multicut* graph partitioning)

* Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavoue, Thomas Brox, Bjorn Andres; *Efficient Decomposition of Image and Mesh Graphs by Lifted Multicuts*; Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1751-1759

Framing vertex finding as a clustering problem

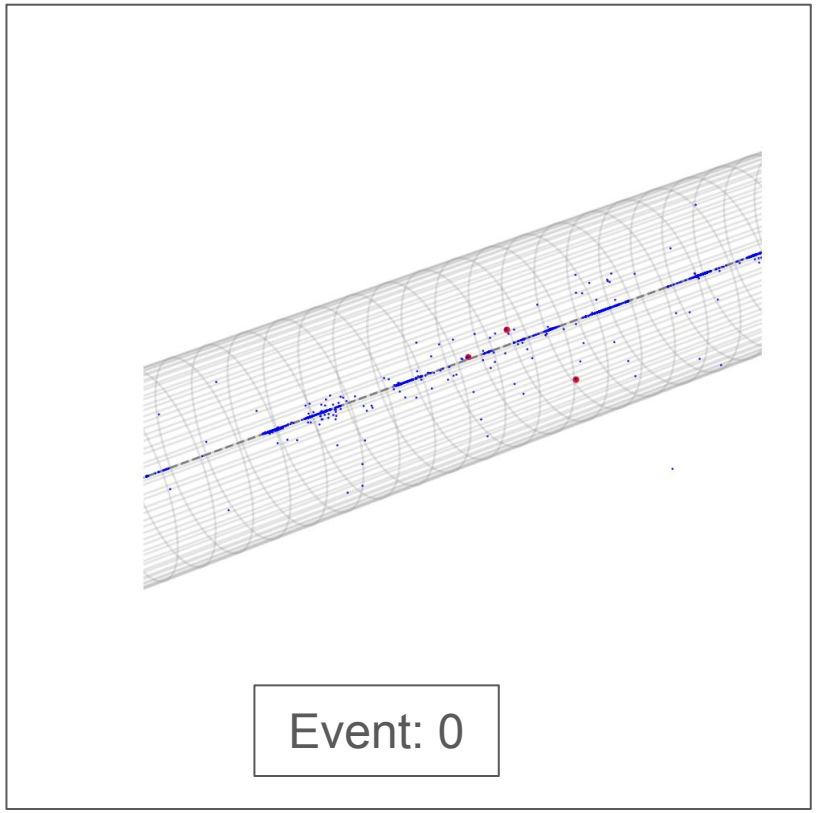


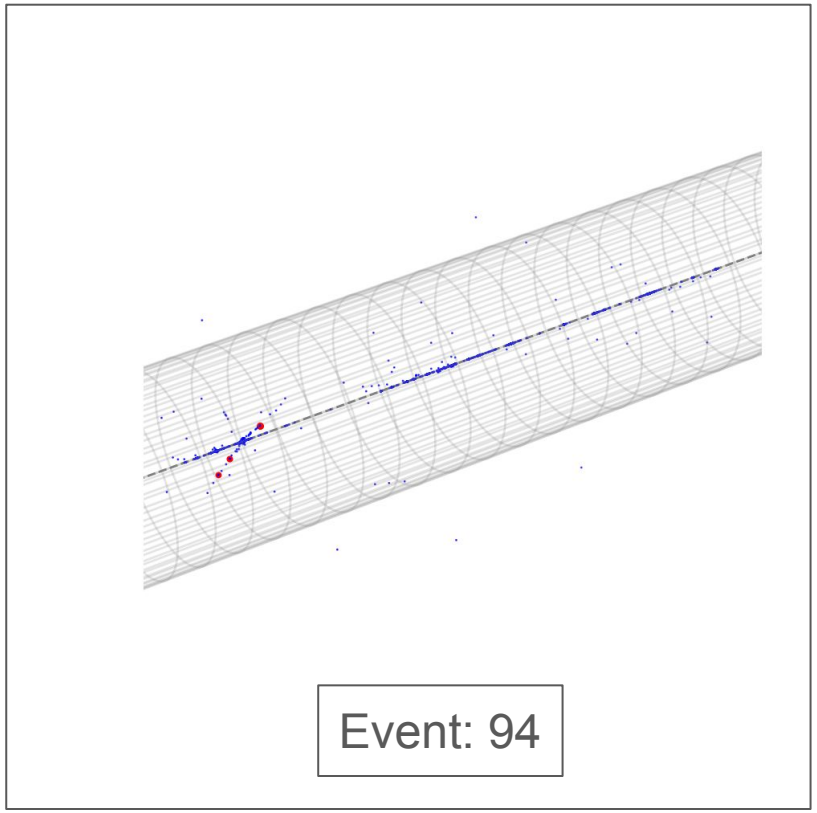
- Each track is a node.
- Each edge is a two-track vertex from Billoir fit.



- Each node is a two-track vertex from Billoir fit.
- Point Cloud based on v**R**.

Visualizing vertices clusters in ODD

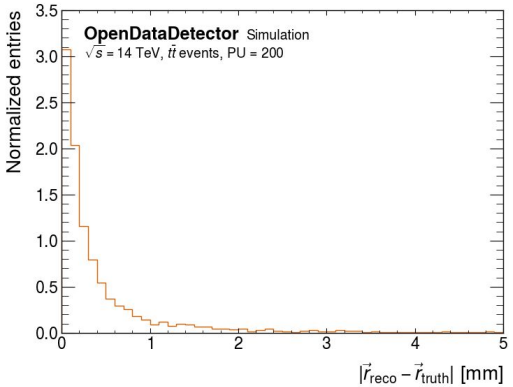


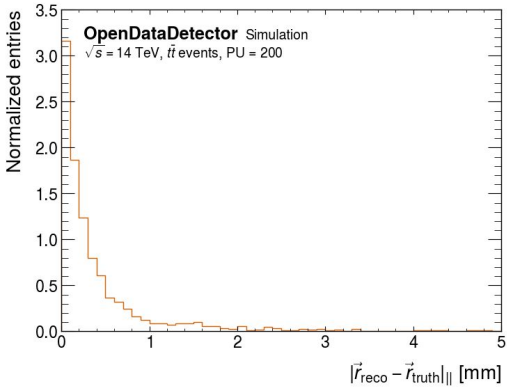


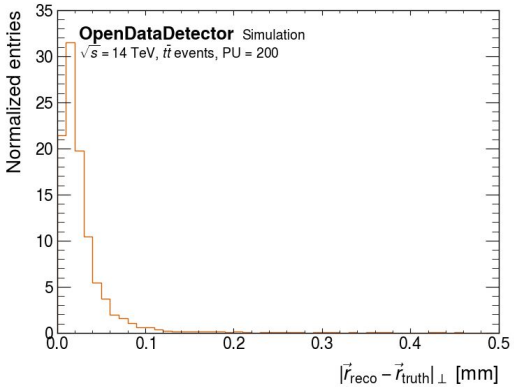
Performance metric for vertex finding

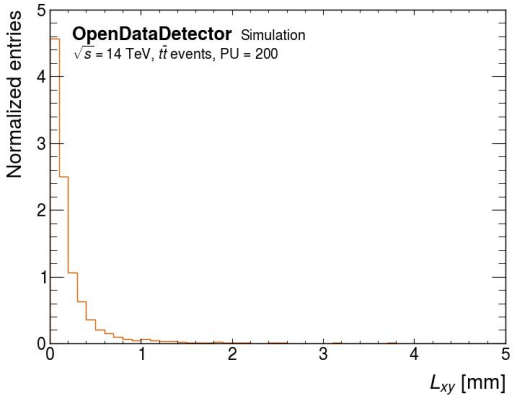
- Strategy for truth-matching:
  - ➢ At least 50% of the tracks from a truth vertex need to be associated with the reco vertex.
  - ➢ If multiple truth vertices match with a single reco vertex, the one with higher number of *common* tracks is chosen.
    - ■ If ambiguity still persists, the one closest to the cluster-centre is chosen.
- For measuring the performance of the clustering method:
  - ➢ Only the truth vertices with 3 or more tracks have been considered.
    - ■ The fraction of such vertices matched with a reco one is a metric for clustering.
- For measuring the performance of “Billoir fast vertex fit” itself:
  - ➢ Only the two-track truth vertices have been considered.
    - ■ For truth matching, **both** tracks are required to be associated with the reco one.

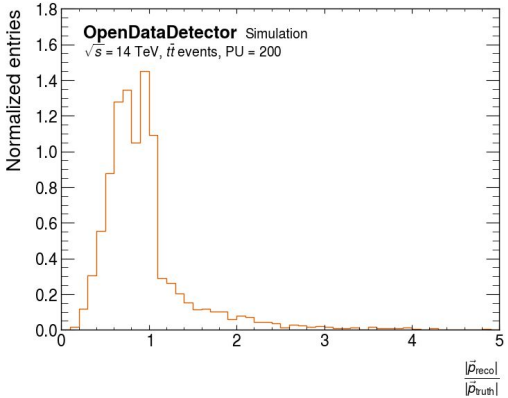
Performance of Point Cloud clustering







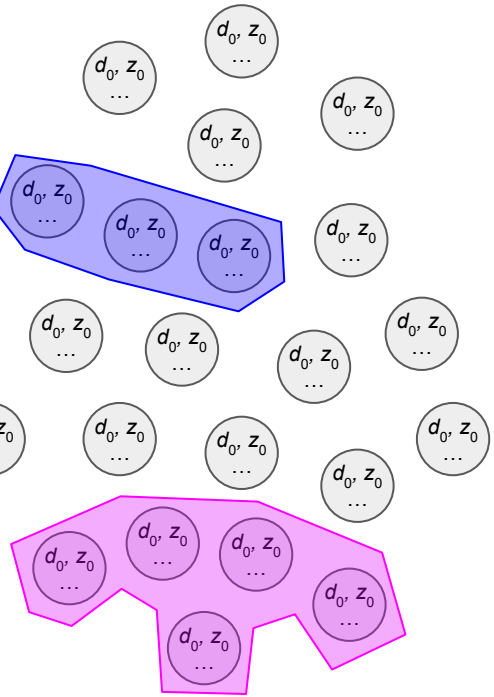




From Point Cloud to HyperGraph

- Point Clouds (of two-track vertices) are nice for Physical intuition, providing a simple bounding-box heuristic enabling us to use any clustering method.
- But they have a significant caveat: Individual track features are lost!
  - ➢ No way to improve upon Billoir fit for estimating track-pair compatibility.
    - Theoretically one can append the features of the two tracks to the vertex-level features.
      - But this looks like more like a hack (might work nevertheless).
- Let's try to find some alternative way to represent the problem.
  - ➢ Hypergraph: Generalization of graph, but with hyperedges.
    - Hyperedge: Generalization of edge, connecting two or more nodes.
      - Secondary vertices can be represented as hyperedges.

From Graph (of tracks) to HyperGraph



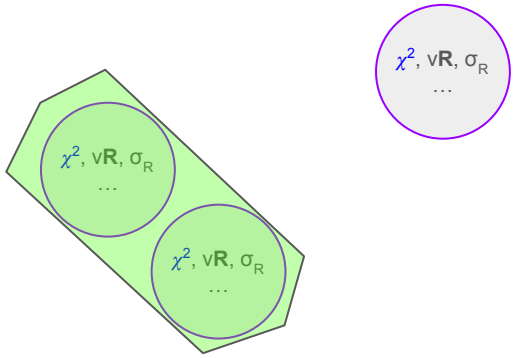
What happens to these nodes?

- Does each of them get its own “partition”?

This issue disappears as soon as we frame it as a **HyperEdge prediction** problem:

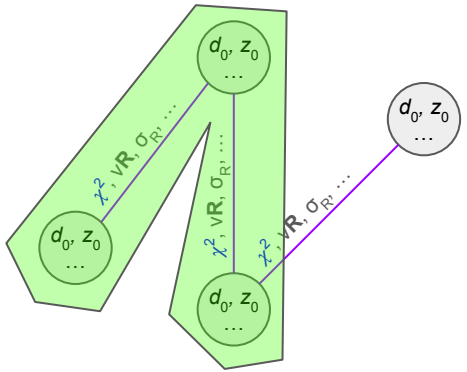
- A hyperedge can connect any number of nodes.
- A node can belong to multiple hyperedges.
- There can be isolated nodes.

From Point Cloud to HyperGraph



Combine two nodes to form a “cluster”.

➢ Using some node feature(s), e.g. v**R**.

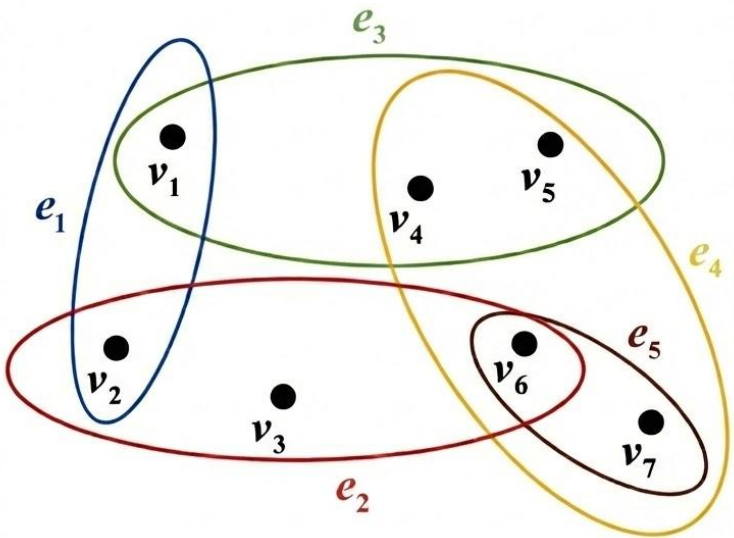


Combine two edges to form a HyperEdge.

➢ Using some edge feature(s), e.g. v**R**.

v**R** refers to the position of the fitted two-track vertex in the 3D space.

HyperGraph representation: Incidence matrix



	e_1	e_2	e_3	e_5
v_1	1	0	1	0
v_2	1	1	0	0
v_3	0	1	0	0
v_4	0	0	1	0
v_5	0	0	1	0
v_6	0	1	0	1
v_7	0	0	0	1

H

Incidence matrix

	d_0	z_0	θ	ϕ	q/p
v_1					
v_2					
v_3					
v_4					
v_5					
v_6					
v_7					

X

Features matrix

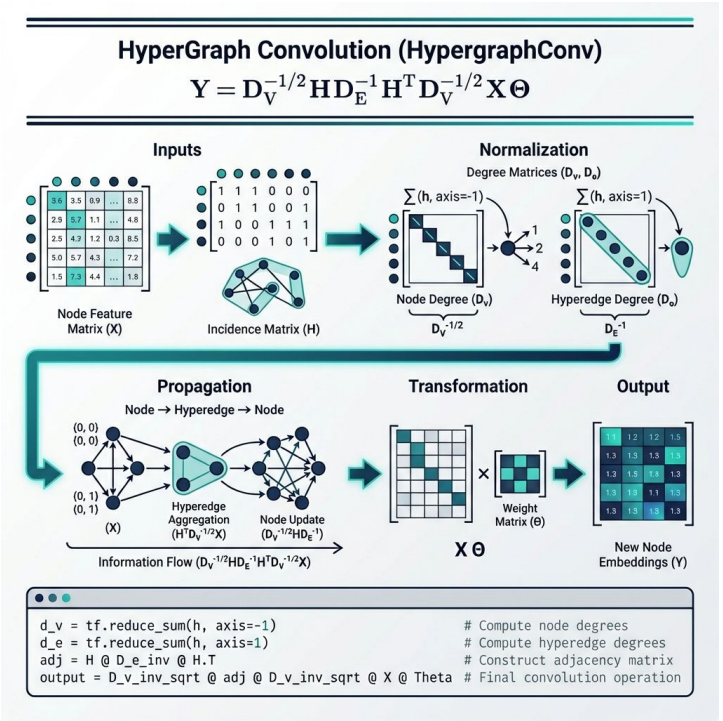
$$H_{ij} = \begin{cases} 1, & \text{if } v_i \in e_j \\ 0, & \text{otherwise.} \end{cases}$$

Message passing on HyperGraph: generalization of GCN

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{D}_V^{-1/2}\mathbf{H}\mathbf{W}\mathbf{D}_E^{-1}\mathbf{H}^T\mathbf{D}_V^{-1/2}\mathbf{X}^{(l)}\Theta\right)$$

Where:

- $\mathbf{X}^{(l)}$ is the input node features.
- $\mathbf{H}$ is the **Incidence Matrix** (defining the hypergraph structure).
- $\mathbf{D}_V$ and $\mathbf{D}_E$ are degree matrices for vertices and hyperedges.
- $\mathbf{W}$ is the hyperedge weight matrix (assumed to be Identity $\mathbf{I}$ in this specific code implementation).
- Θ is the learnable weight matrix (the filter).



Message passing on HyperGraph: The Clique Expansion

$$\mathbf{A} = \mathbf{H}\mathbf{D}_E^{-1}\mathbf{H}^T$$

Intuition:

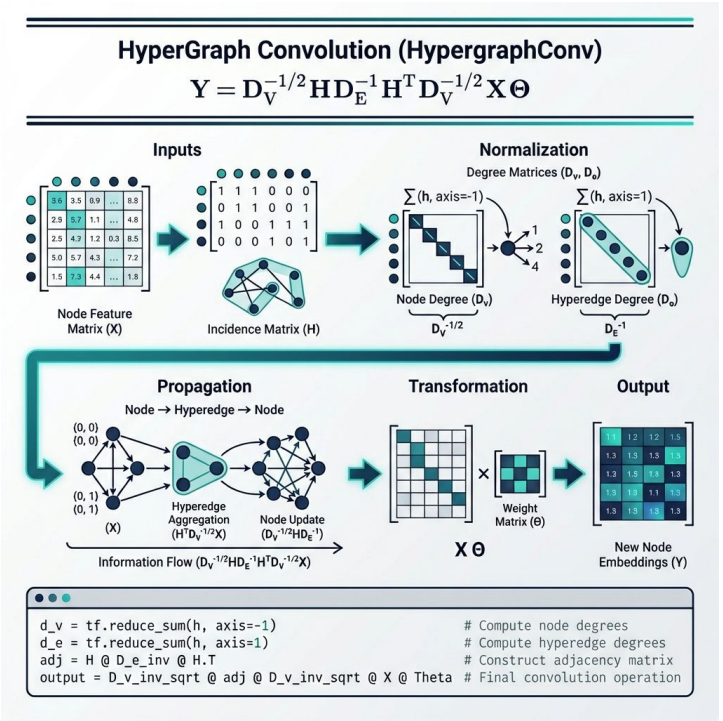
1. $\mathbf{H}^T\mathbf{X}$: Aggregates information from Nodes $\rightarrow$ Hyperedges.
2. $\mathbf{D}_E^{-1}$: Normalizes this information by the size of the hyperedge (averaging).
3. $\mathbf{H}(\dots)$: Distributes information from Hyperedges $\rightarrow$ Nodes.

This effectively converts the hypergraph into a weighted graph where two nodes are connected if they share a hyperedge, weighted by the size of that hyperedge.

Symmetric normalization: $\hat{\mathbf{A}} = \mathbf{D}_V^{-1/2}(\mathbf{H}\mathbf{D}_E^{-1}\mathbf{H}^T)\mathbf{D}_V^{-1/2}$

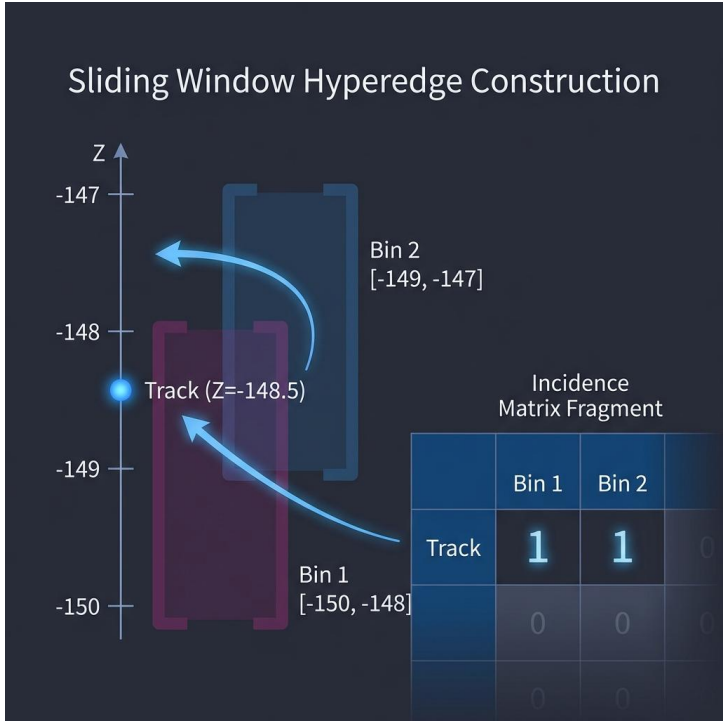
$\mathbf{Y} = \hat{\mathbf{A}}\mathbf{X}\Theta + \mathbf{b}$ ($\mathbf{X}\Theta$ can be replaced with a DNN)

$\mathbf{X}' = \sigma(\mathbf{Y})$



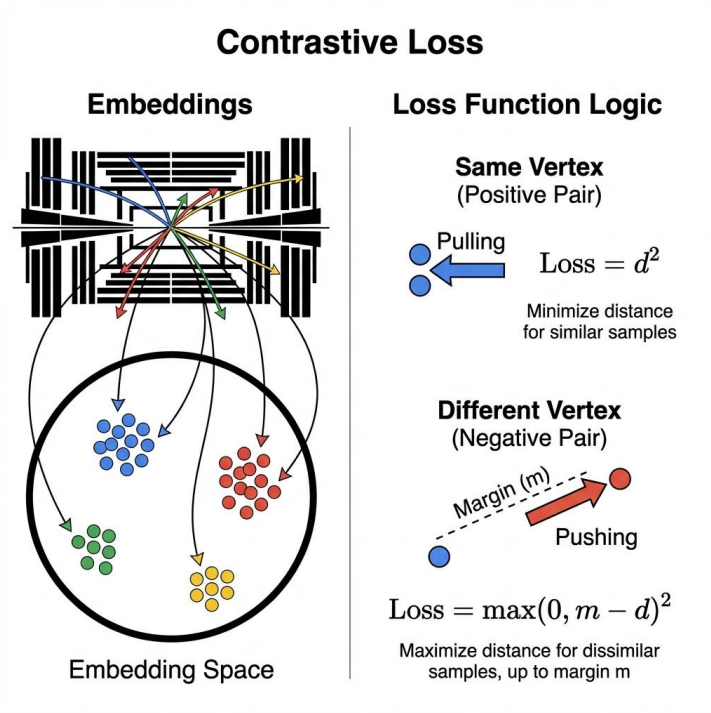
The SV finding algorithm using HyperGraph convolution

- Start with a HyperGraph of tracks as nodes.
  - ➢ Initial incidence matrix is formed by z_0 binning.
  - ➢ Message passing through the hyperedges:
    - Updates the node features.
    - Updates the incidence matrix.
- Final output: Two possibilities
  1. The model outputs the final incidence matrix.
     - Can be immediately used to get the SVs.
     - Constructing the loss function is a bit tricky.
  2. The model transforms the node features to an abstract vector space where clustering is possible.
     - Contrastive loss can be used to train this.



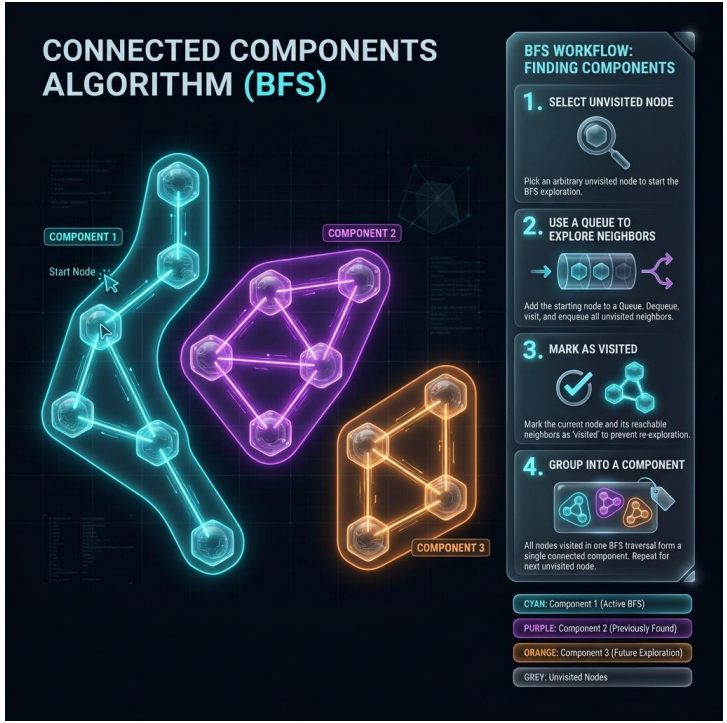
Clustering tracks using *contrastive loss*

- The HyperGraph model transforms the node features into 16-dim vectors.
- The model learns a *transformation*, which:
  - ➢ Minimizes the (Euclidean) distance among the nodes (tracks) from the same cluster (vertex).
  - ➢ Maximizes the distance between two nodes from different clusters, upto a margin m (here, m=1.0).
- Any standard clustering algorithm can be used to find the clusters.
  - ➢ A simple *connected components* algorithm, implemented as a BFS, has been used for this.



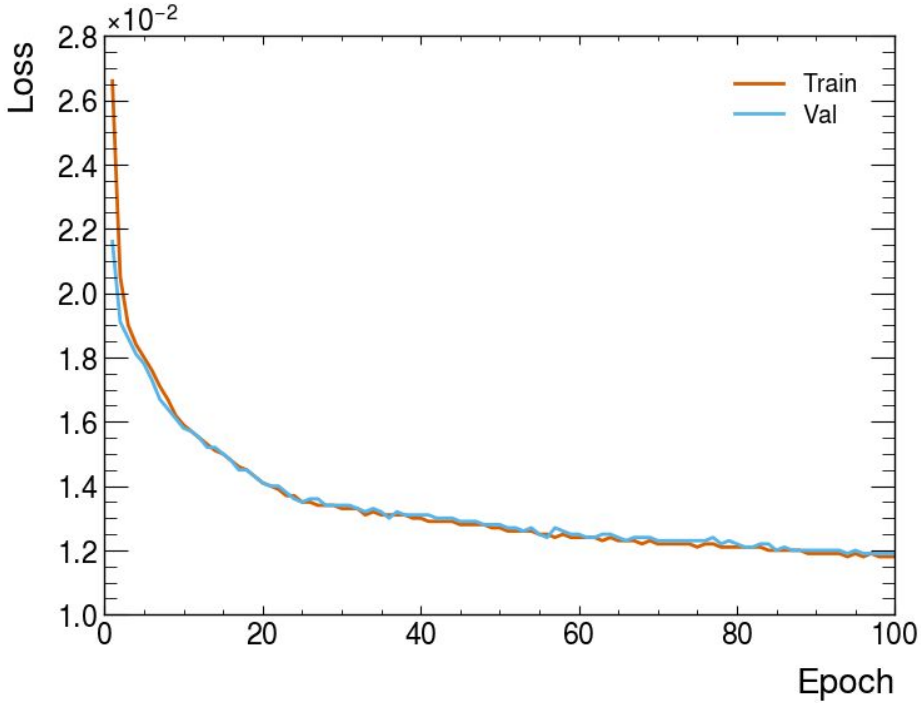
Finding the clusters

- The following simple steps has been used to find the clusters:
  1. The model outputs a 16-dim vector for each node.
  2. Calculate pairwise distances in that 16-dim space.
  3. Put a threshold (0.5, as m=1.0) on those distances to calculate the (boolean) adjacency matrix.
  4. Apply the *connected components* algorithm.
- DBSCAN could also be used for this.
  - ➢ Although it would likely be an overkill.



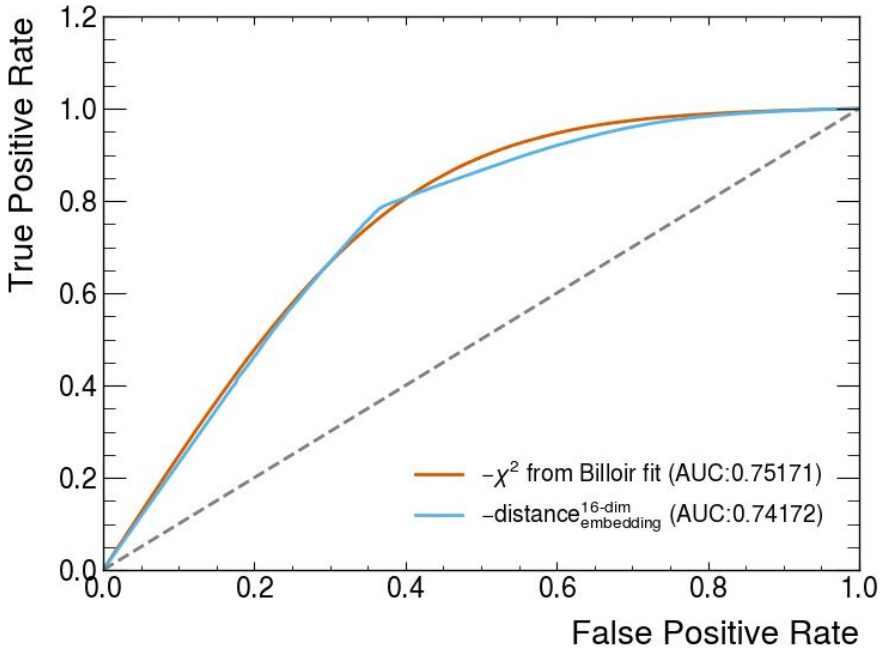
Training setup

- Using a dataset of 6381 events.
  - ➢ Simulated ttbar events using Pythia8 and Geant4 (for ODD with ACTS).
  - ➢ 80:20 train/val split.
- Sequential model:
  - ➢ Layers:
    - HypergraphConv(64, "relu")
    - HypergraphConv(32, "relu")
    - Dense(16, activation=None)
  - ➢ Input padding:
    - MAX_TRACKS = 1000
    - MAX_VERTICES = 200



Performance

- For initial performance estimate:
  - ➢ The model is compared with Billoir fit for track-pair compatibility task.
    - A binary classification task.
  - ➢ Performance of this HyperGraph model is already similar to Billoir fit.
- Points to note:
  - ➢ The HyperGraph model doesn't use any information from Billoir fit.
  - ➢ Uses only 5 track features: d_0, z_0, θ, ϕ and q/p.



Summary and outlook

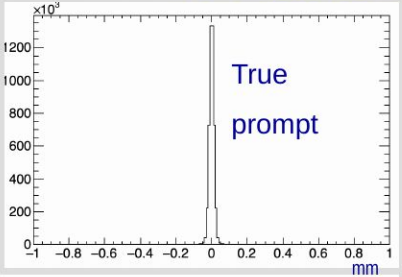
- Secondary vertexing at HL-LHC can be formulated in two different ways:
  1. Simultaneous primary and secondary vertex finding.
     - Computationally expensive, but can be readily framed as a graph-partitioning problem.
     - Might be susceptible to fake or impure (mixed) vertices.
  2. Standalone secondary vertex finding.
     - Significantly less computational complexity, but needs a way to treat isolated nodes.
     - Might suffer from less efficiency to find SVs close to the beam line.
- A preliminary convolution-based HyperGraph model has been developed.
  - ➢ Currently uses *contrastive loss* to map the tracks to a low-dim space.
    - … where the SVs can be found as clusters.
  - ➢ Can be modified to directly predict the vertices (hyperedges) as incidence matrix.
  - ➢ Alternate methods of message passing (e.g. attention based) will be investigated.

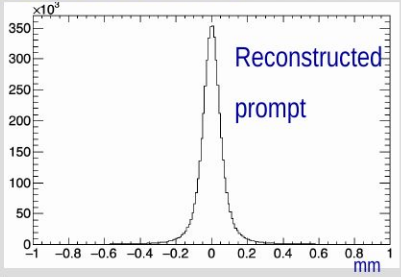
Backup

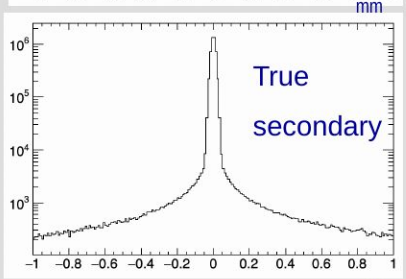
SV challenge

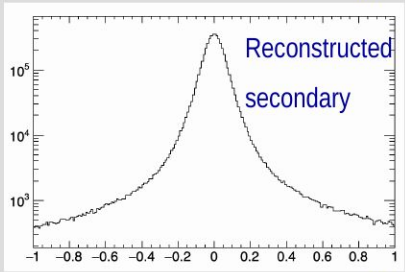
Universität Siegen

Track impact parameters wrt the beamline

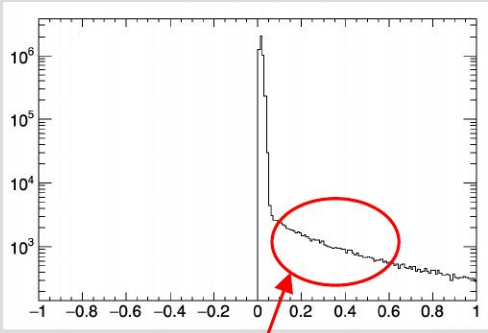






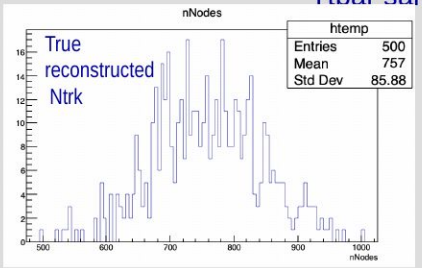


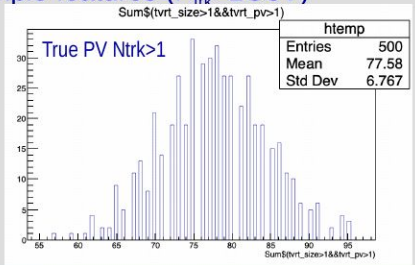
True R of vertices

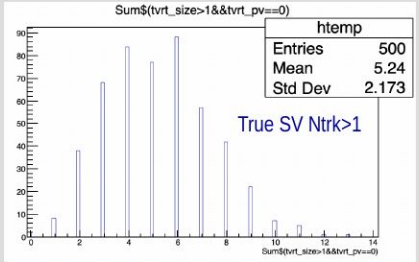


Problematic region for PV/SV reconstruction due to resolution (~equivalent to image blurring?). Hoped to resolve using ML.

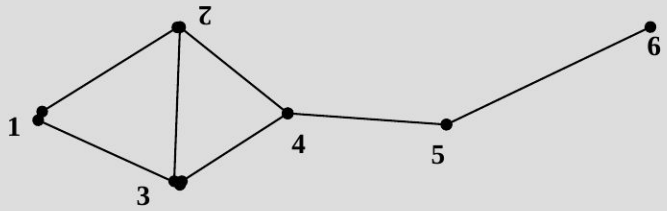
Ttbar sample features (P_{trk}>1GeV)







Edge weights with DGL



Example:
6 tracks (nodes)
7 possible 2-track vertices (edges)

Node vector of features(track parameters): td0, tZ, tPhi, tEta, tQoP, tTheta, tChi2, tNDoF, tCovD0, tCovZ,tCovD0Z, tSignif

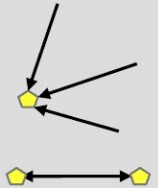
Edge vector of features(2-track vertex): vChi2, vX, vY, vZ, vcovXX, vcovXY, vcovYY, vcovXZ, vcovYZ, vcovZZ, vsumPt, vEta, vPhi, vDZ, massPiPi, vtrue, vCharge, isGamma, isKs, isLambda + truth_label

Edge weights estimation is implemented in DGL

1) Node_hidden_state = NN{Concat(Node_features, Mean(Edge_features)}

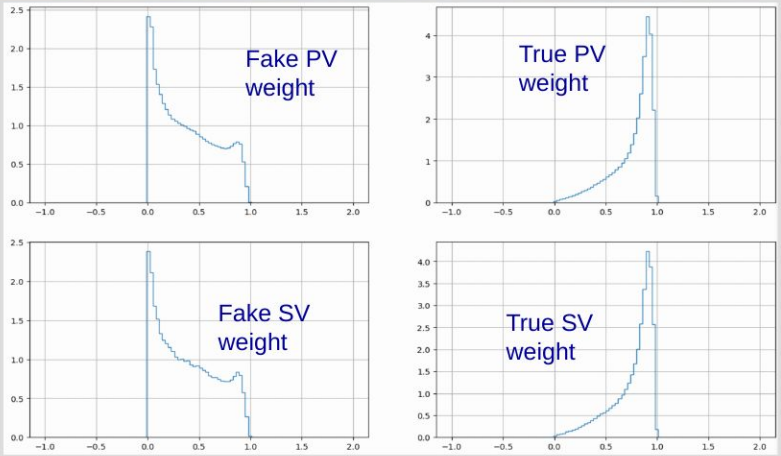
2) Edge_weight = NN{ Concat(Node_hidden_state_i, Edge_features, Node_hidden_state_j)}

3) Node_hidden_state = NN{Concat(Node_features, Weighted_Mean(Edge_features)}
4) Upd_Edge_weight = NN{ Concat(Node_hidden_state_i, Edge_features, Node_hidden_state_j)}



Loss – binary cross-entropy (classification)
NN activations – Mish + Sigmoid to get probability

DGL results (ttbar 1000ev)



AUC PV= 0.8472 AUC SV= 0.8403

For comparison - XGBoost classification of edges based on the same edge features: AUC ~ 0.746

Exploit Minimum Cost Multi-Cut algorithm to cluster track compatibility graphs with GNN-weighted edges

LMP weight	Ncl 1-track	Ncl N-track	effic. SV	purity SV	mixed 2trk clst	Lost sec.track	VI PV	ARI PV	VI SV	ARI SV
GNN wgt	190	54	9%	60%	2.1	49%	2.33	0.443	0.76	0.112
Ideal	65	83	100%	100%	0	0	0	1	0	1

Statistical metrics: Variation of Information(VI), Adjusted Rand index (ARI) are calculated separately on prompt track (PV) and secondary track (SV) collections