Secondary Vertexing using HyperGraph

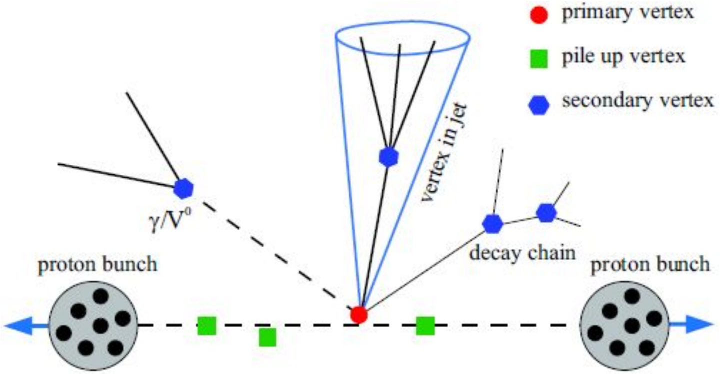
Diptaparna Biswas

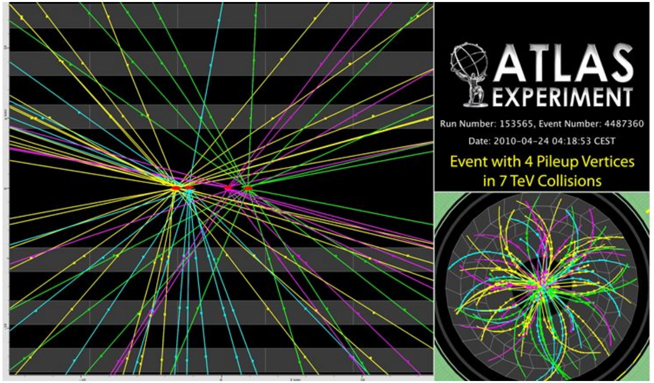
Universität Siegen / HEP Experimental High Energy Physics CPPS Center for Particle Physics Siegen

STAR Sensing and Sensibility Transcending Disciplines for a Responsible Future

TrackOpt meeting
16th January 2025

Recap: Primary vertexing vs. secondary vertexing



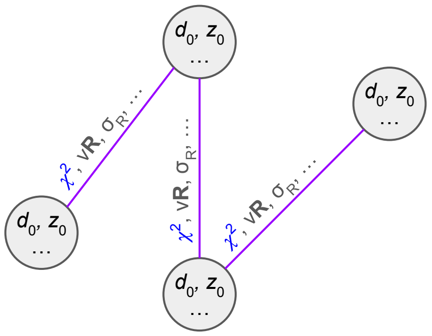


- For our purpose, we call both “primary vertex” and “pileup vertex” inclusively as the primary vertices.
- Each track in an event is associated to one of these primary vertices.
  - ➢ This includes tracks originating from the secondary vertices.

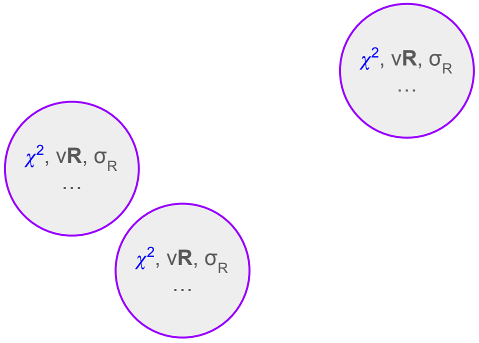
Recap: Framing vertex finding as a clustering problem

- 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.
- New proposal: 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)

Recap: Framing vertex finding as a clustering problem

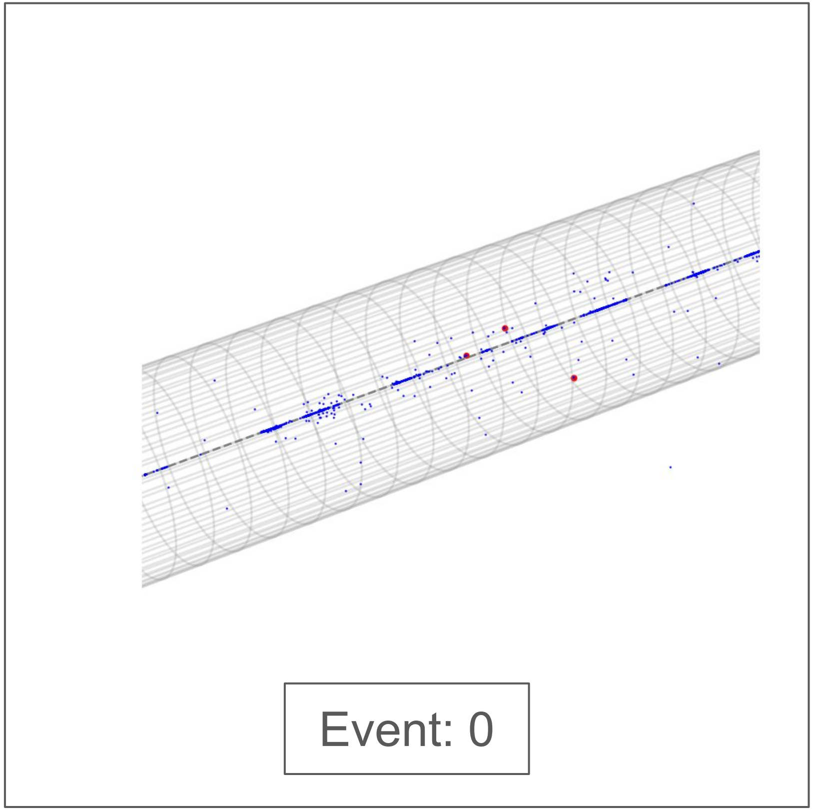


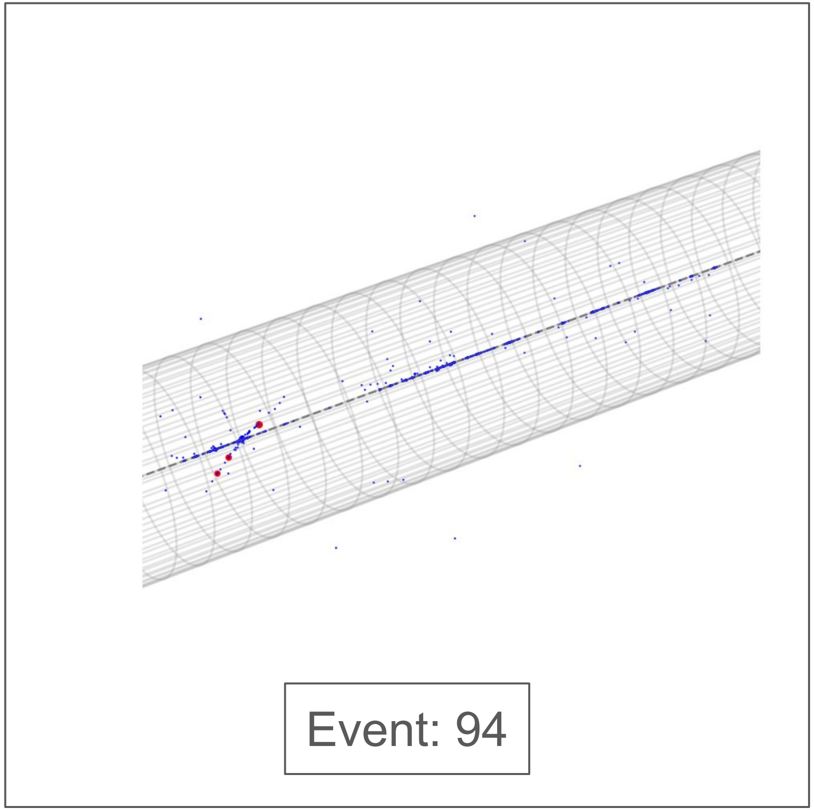
- ➢ Each track is a node.
- ➢ Each edge is a two-track vertex from Billoir fit.



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

Recap: Visualizing vertices clusters in ODD





From Point Cloud to HyperGraph

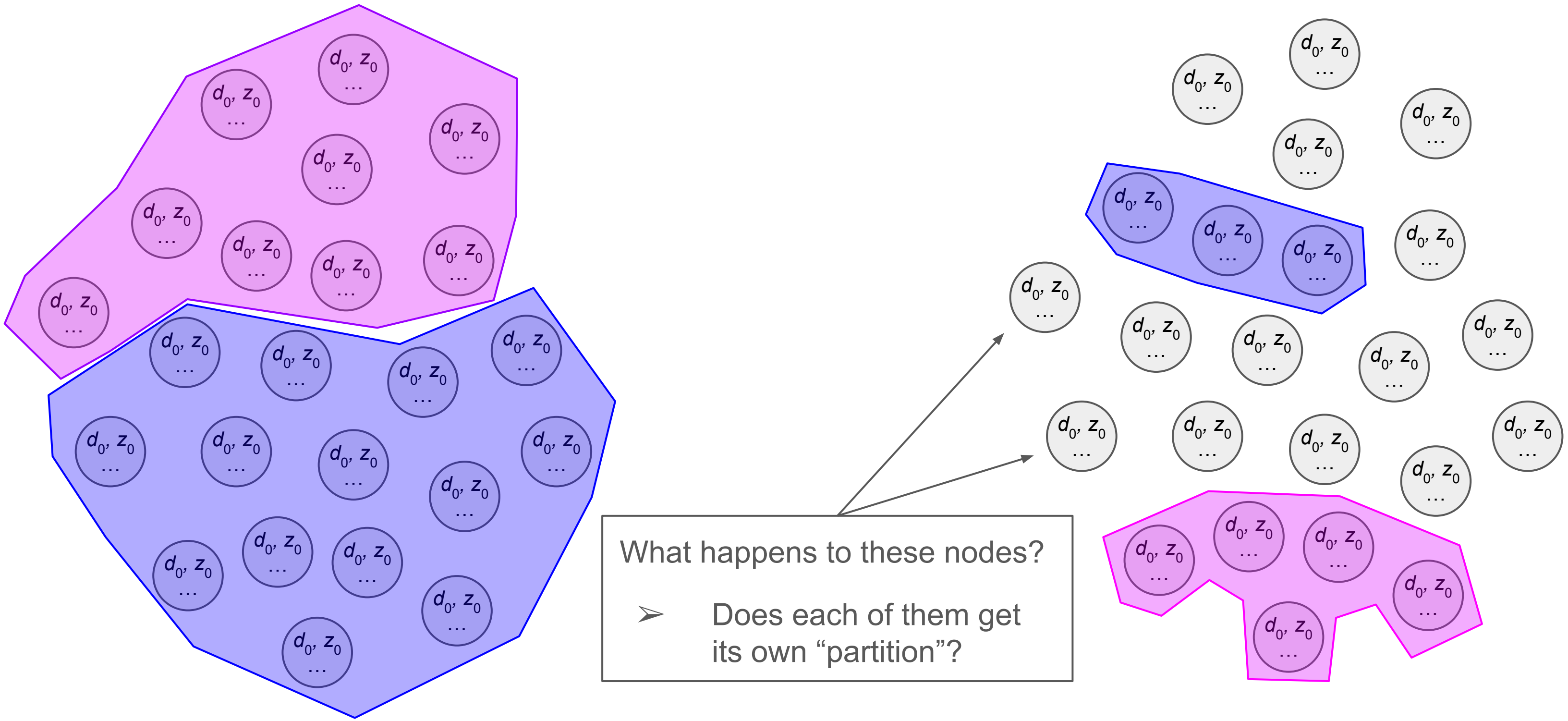
- Point Clouds (of two-track vertices) are nice for Physical intuition, providing a simple bounding-box heuristic enabling us to use any clustering method.
- But they have a significant caveat: Individual track features are lost!
  - ➢ No way to improve upon Billoir fit for estimating track-pair compatibility.
    - ■ 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.

From Graph (of tracks) to HyperGraph

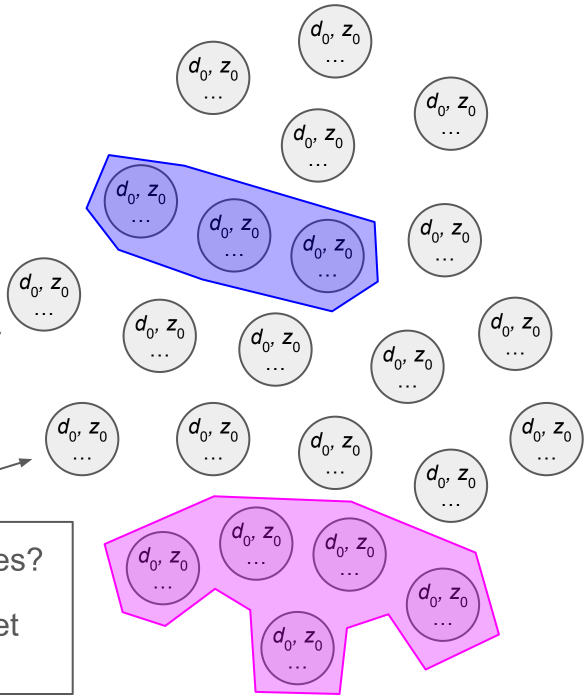
- Secondary vertexing is *very different* from primary vertexing!

Primary Vertexing	**Secondary Vertexing**
Each track in an event belong to one of the PVs.	Most of the tracks in an event doesn't belong to any SV.
A given track always belongs to exactly one PV (after removing ambiguity).	Albeit rare, a given track can belong to multiple SVs.
Suitable to be framed as graph partitioning.	Need to think beyond graph partitioning.

SV finding is not really a graph partitioning problem



From Graph (of tracks) to HyperGraph



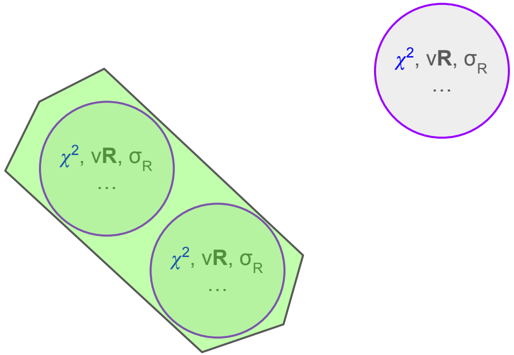
What happens to these nodes?

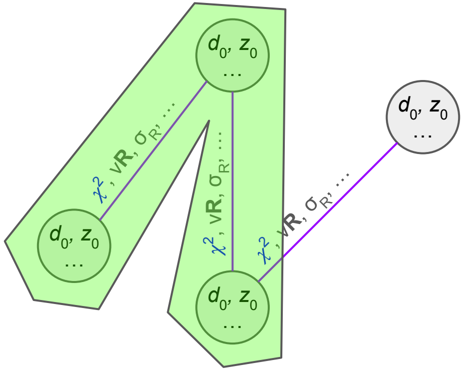
- ➢ Does each of them get its own “partition”?

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





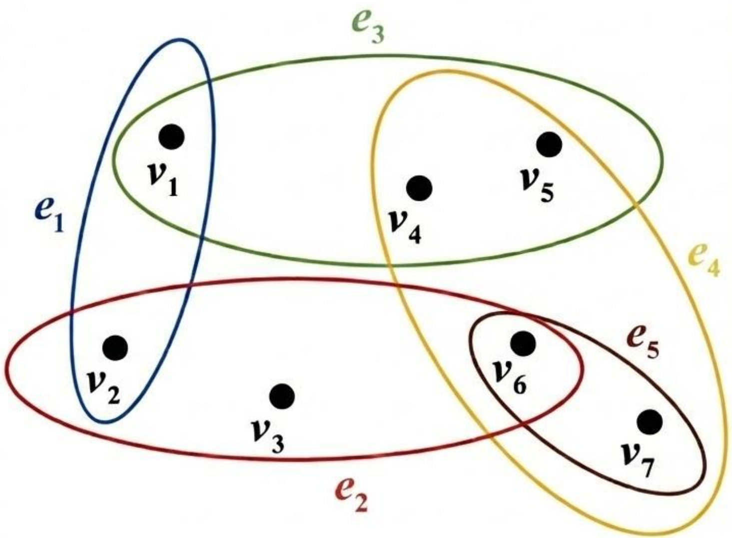
Combine two nodes to form a "cluster".

➢ 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**.

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

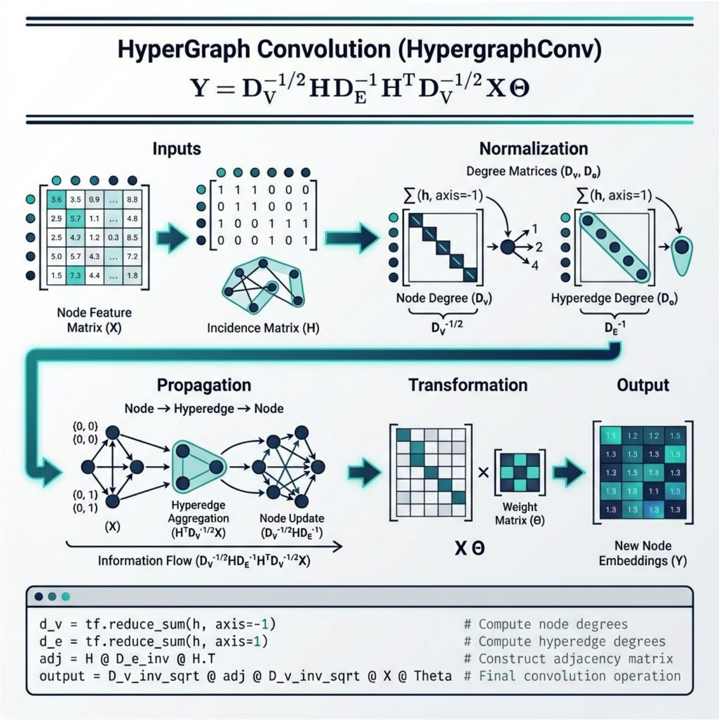
$$H_{ij} = \begin{cases} 1, & \text{if } v_i \in e_j \\ 0, & \text{otherwise.} \end{cases}$$

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:

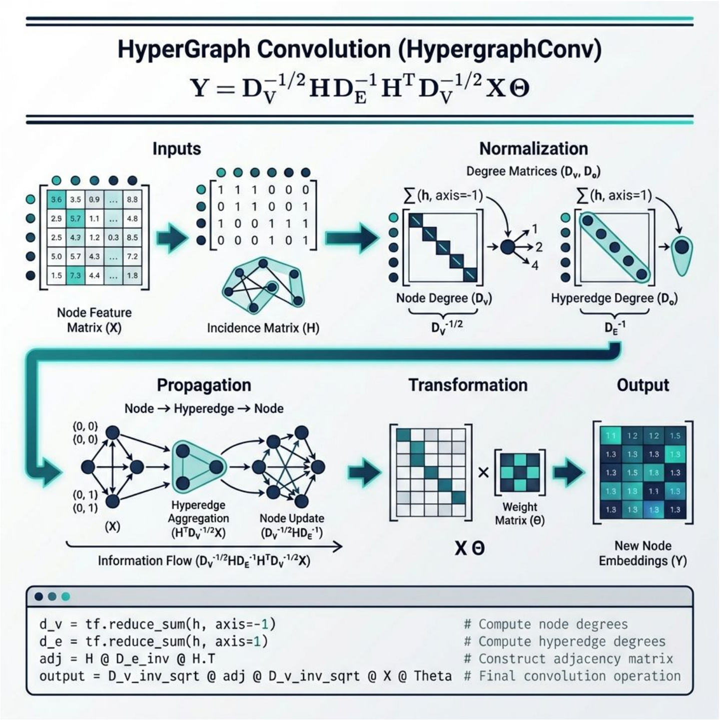
1. $\mathbf{H}^T\mathbf{X}$: Aggregates information from Nodes $\rightarrow$ Hyperedges.
2. $\mathbf{D}_E^{-1}$: Normalizes this information by the size of the hyperedge (averaging).
3. $\mathbf{H}(\dots)$: Distributes information from Hyperedges $\rightarrow$ Nodes.

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)

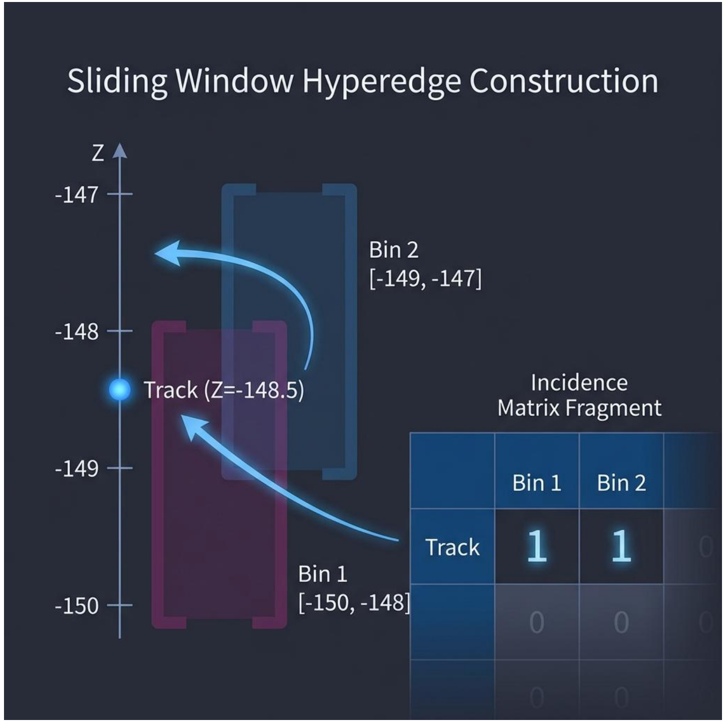
$\mathbf{X}' = \sigma(\mathbf{Y})$



The SV finding algorithm using HyperGraph convolution

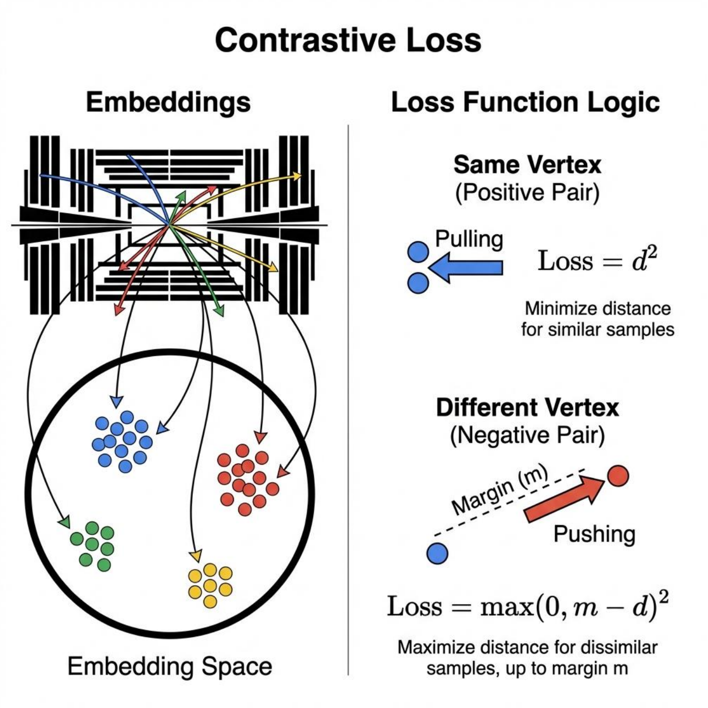
- Start with a HyperGraph of tracks as nodes.
  - Initial incidence matrix is formed by z_0 binning.
  - Message passing through the hyperedges:
    - Updates the node features.
    - Updates the incidence matrix.

- 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.
     - "Construstive loss" can be used to train this.



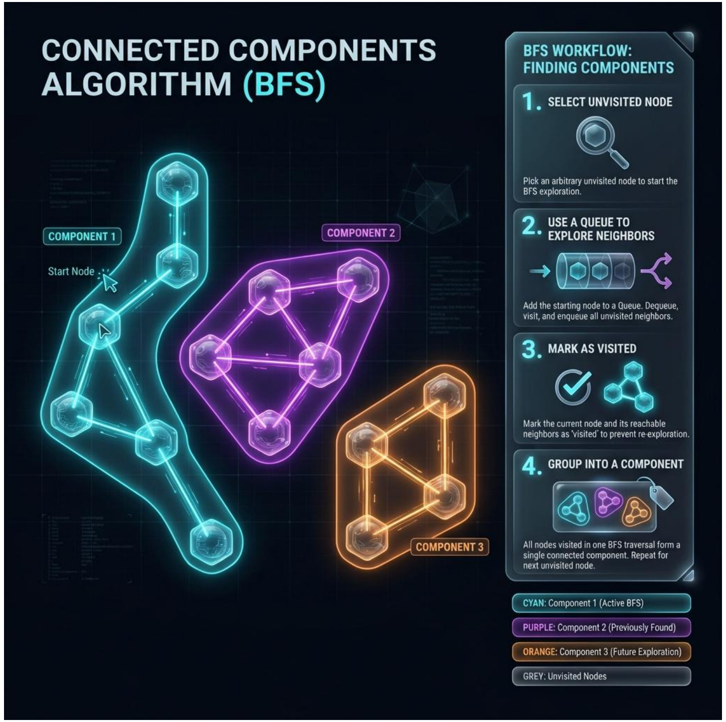
Clustering tracks using “construstive loss”

- 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.



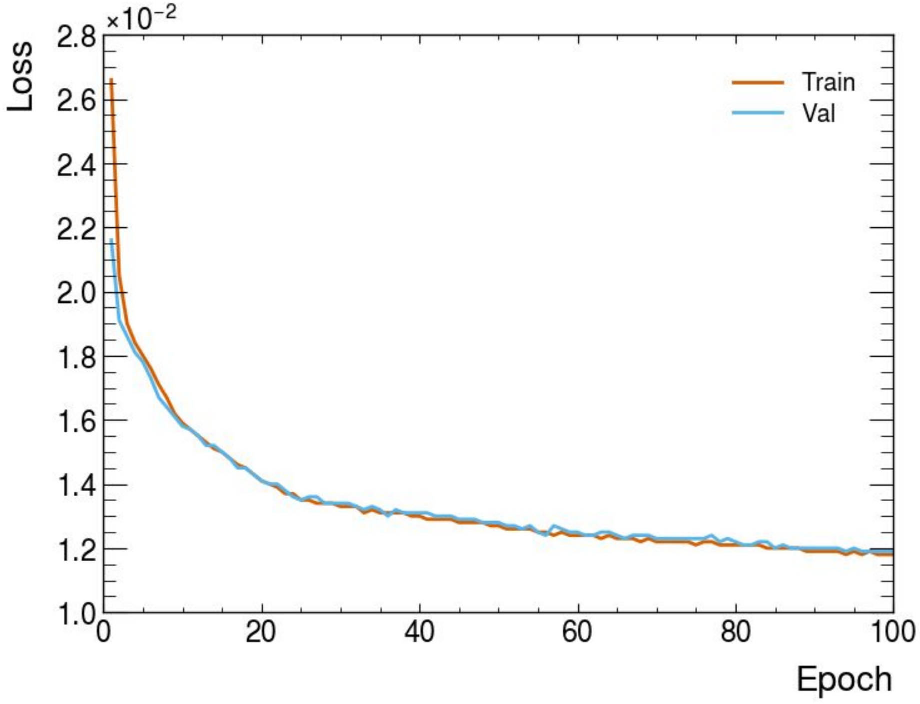
Finding the clusters

- 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.



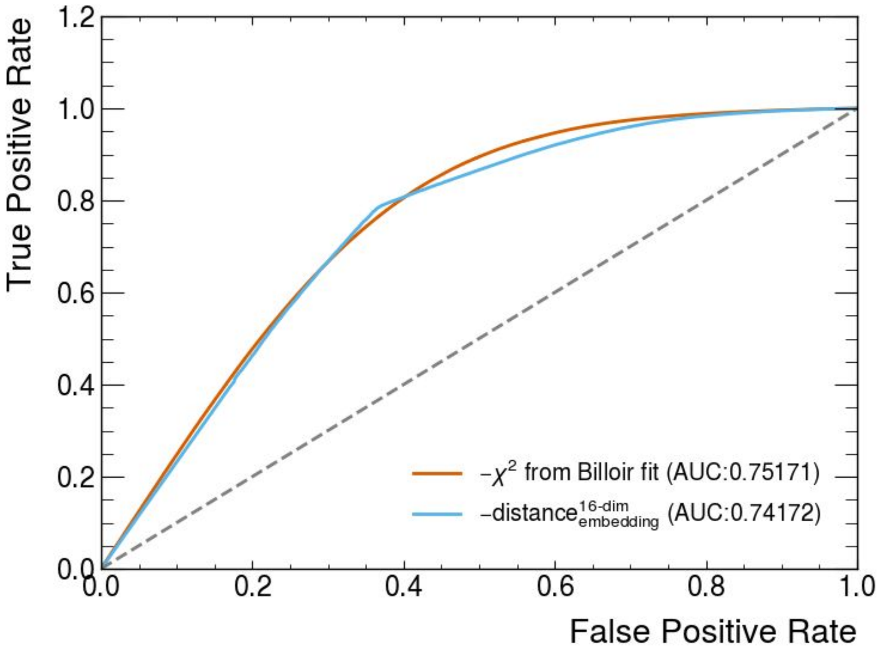
Training setup

- 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



Results

- 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.



Next steps

- Replace the matrix multiplication between the feature matrix and the matrix of learnable parameters (i.e. $\mathbf{X}\Theta$) with a small feed-forward network.
- Use attention mechanism (instead of convolution) for message passing.
- Modify the model architecture and loss function so that the model can directly output the final incidence matrix (i.e. hyperdeges).
- Currently the model uses only 5 track features:
  - ➢ Investigate whether including more features improves performance.
  - ➢ Incorporate the quantities from Billoir fit as model input.

Summary

- Secondary vertexing can be expressed as a HyperEdge prediction task.
  - ➢ Instead of graph partitioning, which is more suitable for primary vertexing.
- A preliminary convolution-based HyperGraph model is developed.
  - ➢ Doesn't use any information from Billoir fit.
  - ➢ Uses only 5 track features as input.
    - ■ As comparison, Billoir fit additionally uses the (co)variances of the 5 track parameters.
  - ➢ Performance is already similar to Billoir fit for track-pair compatibility prediction.
  - ➢ Lots of room for improvement in the HyperGraph model (e.g. applying attention mechanism).
- Currently, the model maps the tracks to a 16-dim embedding space.
  - ➢ Using a constrastive loss, where the secondary vertices are found as clusters.
  - ➢ Can be modified to directly predict the vertices (hyperedges) as incidence matrix.