GNNs for Pattern Recognition and Fast Track Finding

20th April 2023

Nisha Lad (UCL)
Dmitry Emeliyanov (STFC RAL) & Nikos Konstantinidis (UCL)

Outline

- Track Finding & Motivation
- Graph Neural Network (GNN) Algorithm
  - Network Initialization
  - Gaussian Mixture Reduction
  - Neighbourhood Aggregation
  - Track Splitting & Extraction
- Results
  - MC Toy Model
  - TrackML Model
- Conclusions

Track Finding & Motivation

- **Modern HEP Particle Detectors**:
  - Formed from different sub-detectors & arranged in cylindrical concentric layers around the nominal interaction point
  - Uniform magnetic field in the centre of the detector parallel to the beam line
  - The ideal trajectory of charged particles in 3D is a helix (axis parallel to the beam line)
  - **Tracks: reconstructed sequences of hits representing charged particle trajectories**

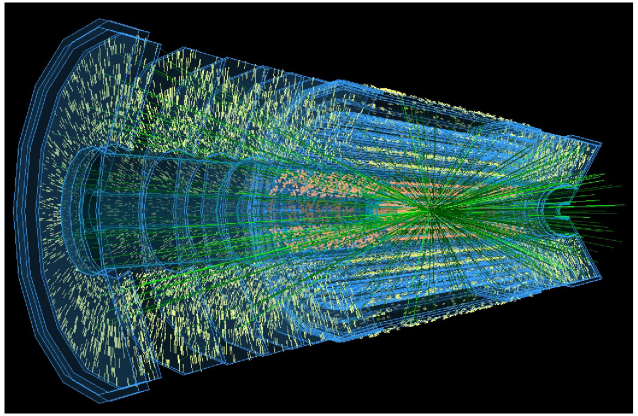
- **Track finding as a Pattern Recognition Problem:**
  - **Associate measurements ('hits') into sequences** representing tracks
  - Scale: O(10^5) hits per event, several 1000s tracks → main consumer of CPU today
  - Current algorithms are based on **combinatorial track following** approaches
  - Future upgrades → increased data from particle detectors i.e. High Luminosity phase of the LHC experiment, reconstruction ~mins with 40 million collisions per second
  - **CPU time increase** creates a huge demand for computing power

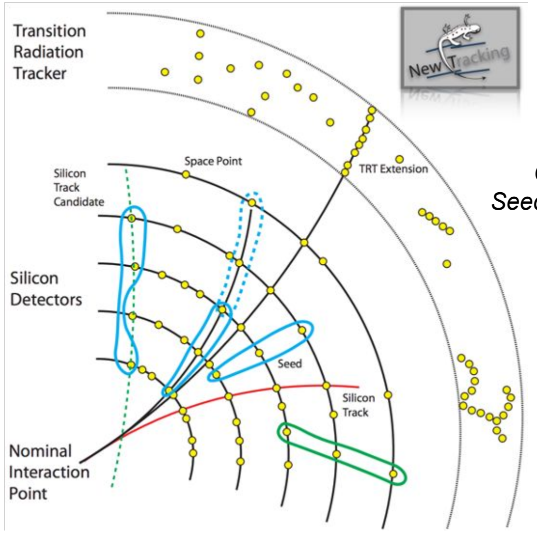
    Motivation for novel approaches in track finding → large savings in CPU

- **Research focus: Inner Detector (ID):**
  - Typically dedicated to track and vertex reconstruction
  - Closest to the beamline & has highest hit occupancy
  - Generally consists of several types of sensors: Pixel & Strips

- **Aim: Explore Track Finding methods utilizing GNNs**
  - Clusters of hits are connected together into a graph network
  - GNN algorithm will serve as a **more sophisticated seeding** for Pixel clusters
  - Preliminary track seeds which can be refined & extended into Strips
  - Such an approach could be **very efficient for saving computation resources**, if the GNN doesn't produce too many fake tracks from random combinations of hits

Simulation of a proton-proton collision expected at the High Luminosity phase of the LHC experiment





Example Combinatorial Seeded Approach

Track Finding on Graph Networks

Overview:

- **Unique method**: unlike traditional approaches where Multi- Layered Perceptrons (MLPs) are trained
- **Architecture: fully iterative pattern recognition** algorithm; at each iteration we discover new track candidates
- **Iteratively resolve ambiguities** by **masking incompatible edge connections** & improves track parameters

How will this be achieved?
Utilizing pattern recognition techniques, we alternate between "edge outlier masking" via ***Gaussian Mixture Reduction (GMR)*** & message passing via ***Neighbourhood Aggregation (NA)*** to iteratively improve the precision of track state estimates. As the network evolves, the edge connections should stabilize.

Algorithm Stages:

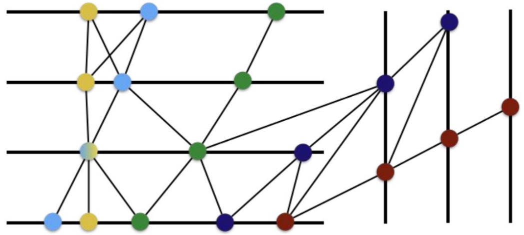
GNN Initialization
→ Out of the box Connected Component Analysis (CCA)

1 *GMR via Clusterization*
Track Extraction ***via KF***

2 *Information Aggregation & **KF Update***
Track Extraction ***via KF***

3 *Update Network State*

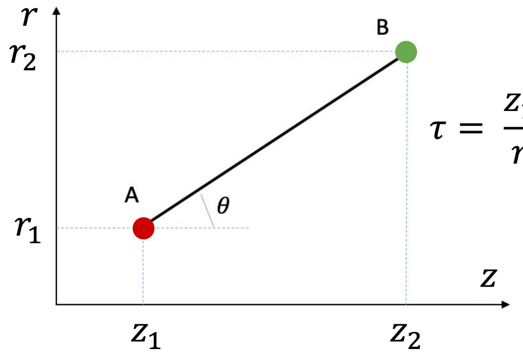
Example GNN illustration



Unique use of Kalman Filters (KF):
Exploits a priori knowledge about charged particle dynamics, by using **simplified KFs as mechanisms for information propagation & track extraction**

GNN Initialization

1 Linear Model (r, z) Plane:



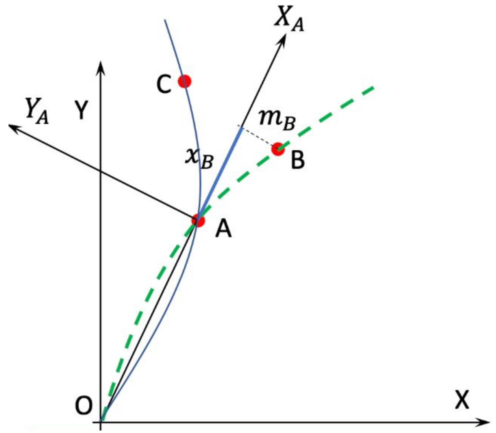
$$\tau = \frac{z_1 - z_2}{r_1 - r_2}$$

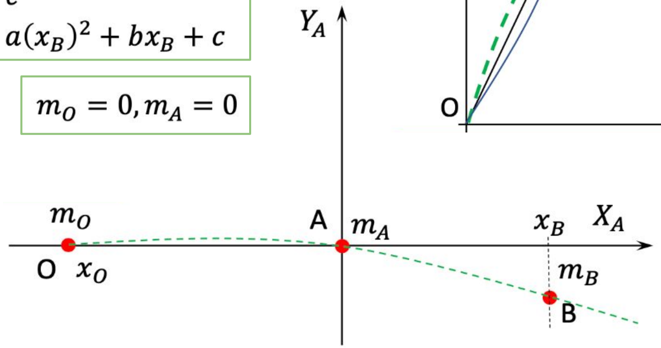
2 Parabolic Model (x, y) Plane:

- m are measurements of track position
- **Local (node specific) coordinate-system**
- Track states at node A: **Parabolas**
- Equations for parabola parameters at node A:

$$\boxed{\begin{aligned} m_O &= a(x_O)^2 + bx_O + c \\ m_A &= c \\ m_B &= a(x_B)^2 + bx_B + c \end{aligned}}$$

- We assume: $\boxed{m_O = 0, m_A = 0}$



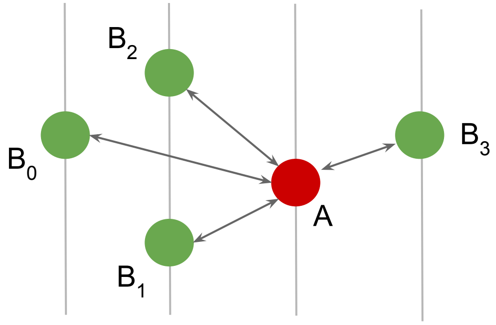


Gaussian Mixture at Each Node:

Given a node A, conditioned on its neighbourhood B_j:

- Compute X_{ij} (track state estimate)
- Compute C_{ij} (edge covariance)
- This forms a ***Gaussian component*** φ_{ij}
- Store all components at node A
- ***Gaussian mixture $g_i(X)$ at node A***

$$\boxed{g_i(X) = \sum_j w_{ij}\varphi_{ij}(X, X_{ij}, C_{ij})}$$



3 Form a Joint Vector State:

$$\hat{X} = [\overbrace{a \quad b}^{\text{Parabolic parameters from (x, y) plane}} \quad \underbrace{\tau}_{\text{Inverse inclination from (r, z) plane}}]$$

Iteration 1: Gaussian Mixture Reduction

Aim: **Identify & mask outlier edge connections**

Why reduce the mixture?

- **Recursively processing**, no. of components & calculations can grow exponentially
- **Solution**: Given a node, **model each edge as a Gaussian component,** forms a **Gaussian mixture** with N components, find a reduced mixture with $M < N$ components, such that some deviation measure is below a threshold

GMRC: Gaussian Mixture Reduction via Clustering

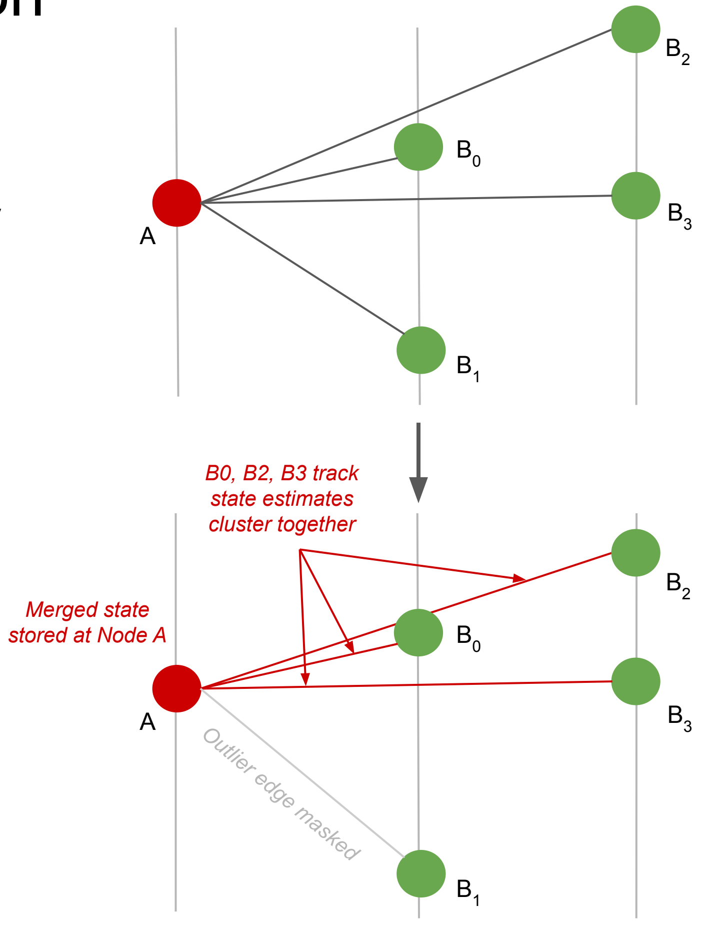
- Can be achieved in two ways:
  - **Iteratively merging** Gaussian components → forms a **merged state**
  - **Pruning components** by **masking outliers**

Pseudocode:

- Based on k means clustering, but with k=1
- Introduce a distance metric between all edge connections - KL divergence
- Compute pairwise distances, start with the smallest:
- If metric < threshold → merge components
- **Progressive convergence**, check all edge pairs until no further merging possible
- Outliers identified & masked

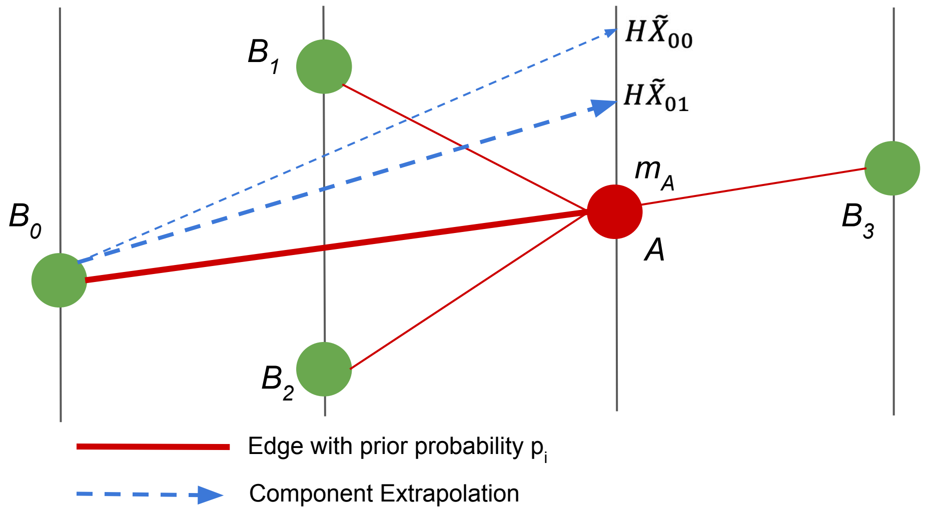
KL (Kullback–Leibler) divergence:

- Deviation measure, serves as a cost function
- Measure of the **distance between 2 probability distributions**
- **Optimal** KL threshold learned using a **Support Vector Machine (SVM) classifier**



A
B_2
B_0
B_3
B_1
B0, B2, B3 track state estimates cluster together
Merged state stored at Node A
A
B_2
B_0
B_3
B_1
Outlier edge masked

Iteration 2: Information Aggregation & Kalman Filter



$$\begin{tikzcd} \boxed{\textit{Message Passing}} \arrow[r] & \boxed{\textit{State Extrapolation}} \arrow[r] & \boxed{\textit{Validation}} \arrow[r] & \boxed{\textit{KF Update}} \end{tikzcd}$$

- **The problem:** given a neighbourhood $\mathcal{N} = \{B_i\}$, calculate the Gaussian mixture representing a possible track state at the node A conditioned upon a measurement m_A
- **The model:** at each neighbour node B_i the track state is represented by a Gaussian mixture, or a reduced mixture (merged state)

$$\boxed{g_i(X) = \sum_j w_{ij}\varphi_{ij}(X, X_{ij}, C_{ij})}$$

- **Information aggregation:** distribute messages to neighbourhood if edge connections are active
- **State Extrapolation:** merged state is extrapolated from neighbour node to node A using linear extrapolation equations
- **Validation:** compute the residual & chi2 distance between extrapolated state and the measurement at node A
- **Perform KF update:** If the chi2 distance is compatible
- Otherwise this edge connection is turned off

Extrapolation:

$$\boxed{\tilde{X}_{ij} = F_i X_{ij}}$$

F_i is the jacobian from B_i to A

Q is the process noise matrix: encompasses all material effects modelled using ***"Ornstein-Uhlenbeck"*** (OU) process

$$\boxed{\tilde{C}_{ij} = F_i \left\{ \sum C_{ij} + Q \right\} F_i^T}$$

Track Splitting & Extraction

Aim: Extract tracks iteratively as the network evolves after each iteration, good track candidates don't need to be processed again

Some remaining networks/isolated nodes will remain as not 100% of ambiguities can be resolved

Criteria for a good candidate:

- ***Candidate must have >= 4 hits: no track fragments***
- ***1 hit per layer****:* no competing/intersecting tracks & no holes
- ***P-val acceptance threshold > 0.01*** (chi2 distance with KF track fit)

Outlier Edges:

- Incompatible edge connections - shown here as greyed out edges in order to split subgraphs and form suitable track candidates - i.e. 1 hit per layer

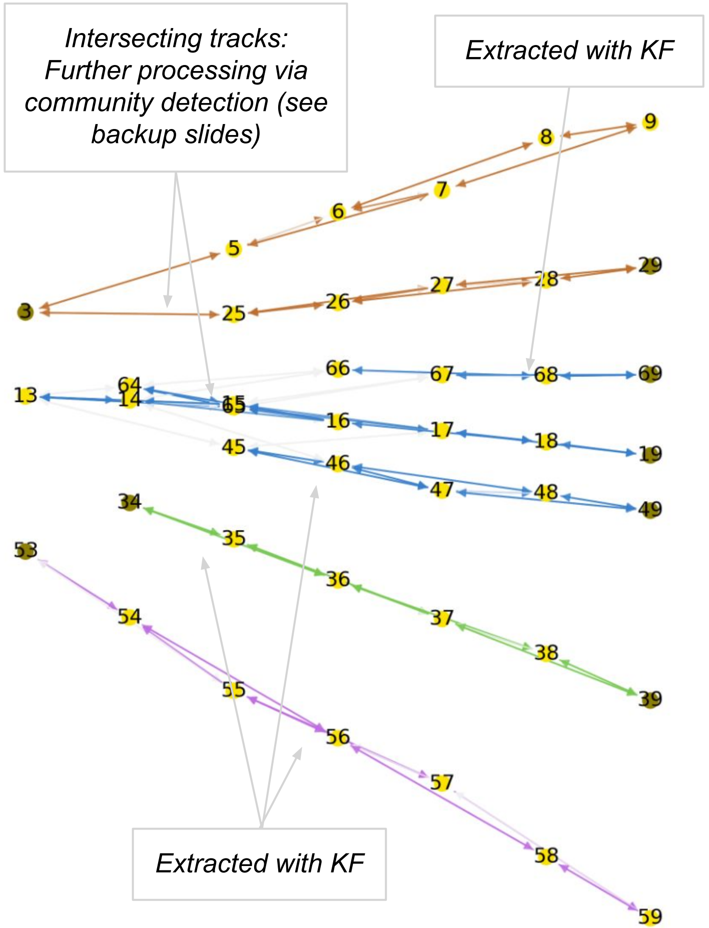
Extracting good candidates:

1. Connected Component Analysis (CCA)

2. Kalman Filter Track Fit

Iteratively predict & update track parameters

3. Community Detection

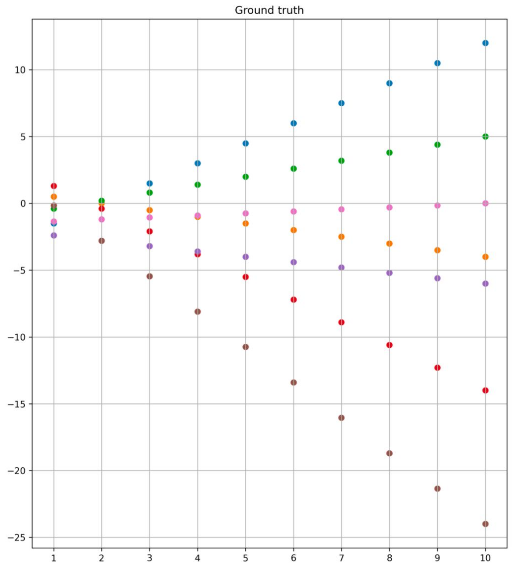


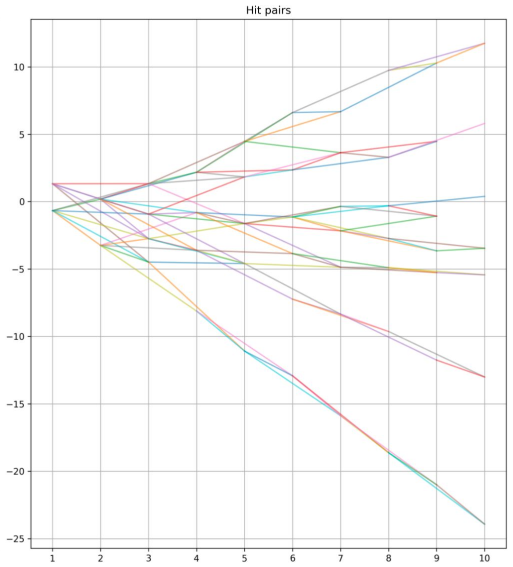
Results

Application on a Simple Simulation

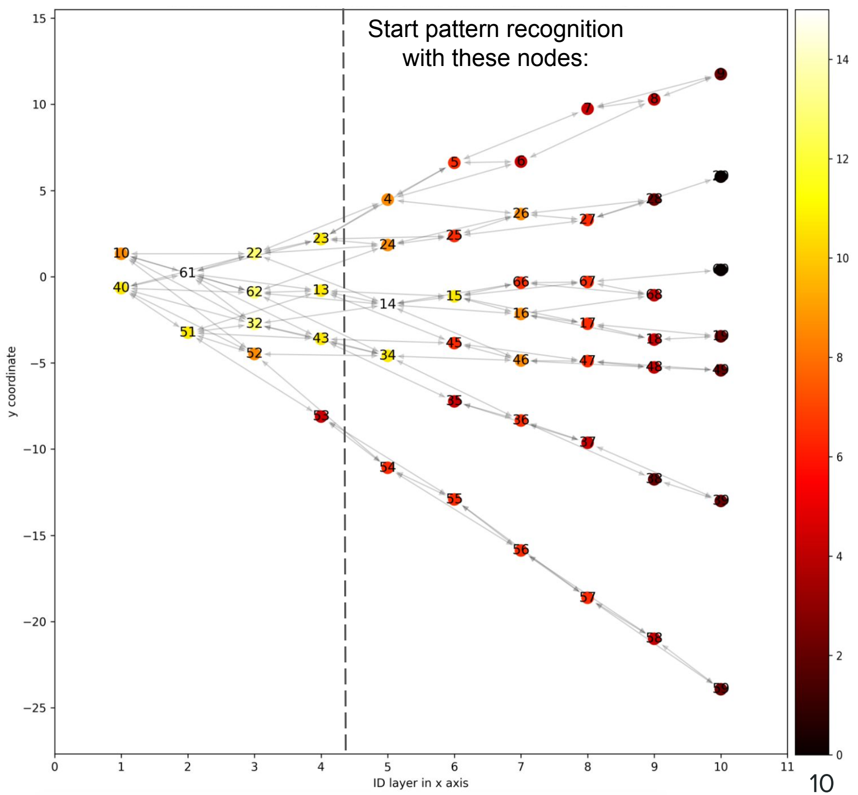
- **Initialize the graph network:**
  - Simple 2D model (70 nodes) using the Python package: **NetworkX**
  - Kalman Filters implemented using Python package: **Filterpy**
  - Nodes as hits & predicted edges using a simple 'hit-pair-predictor'
  - Create a framework that **automatically determines a suitable region for initiating pattern recognition**
  - Criteria: variance of edge orientation

- Remove nodes with high variance
- **Hot nodes (white, yellow):** high variance of edge orientation
- **Cold nodes (orange, red):** low variance of edge orientation
- **Start the pattern recognition in "cold" regions**





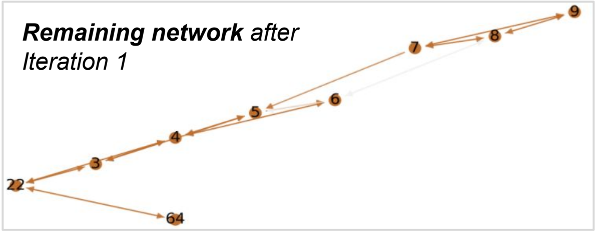
Variance of edge orientation indicating "hot" & "cold" regions



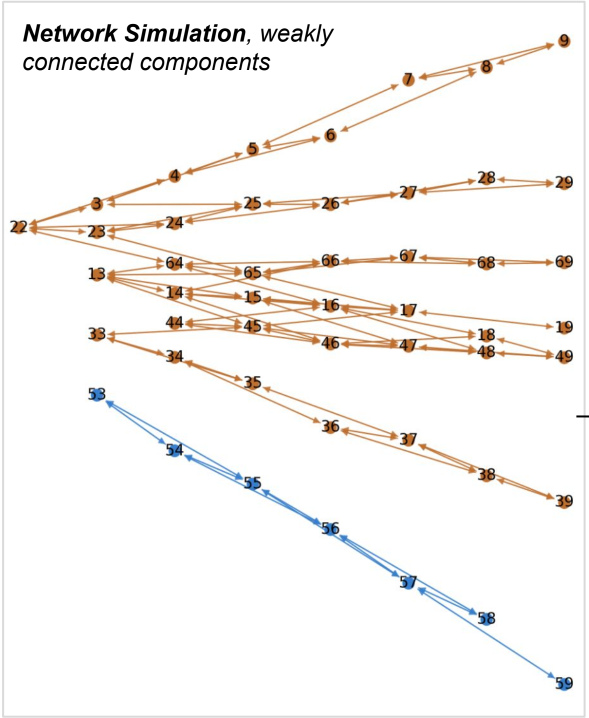
Simple Simulation Results

- Iteration 1 only:
  - 6/7 potential good track candidates are extracted - fast convergence
  - Of which 100% precision with respect to the ground truth
- No further tracks were extracted from remaining network - further processing needed
- **Approximate timings → ~10s**

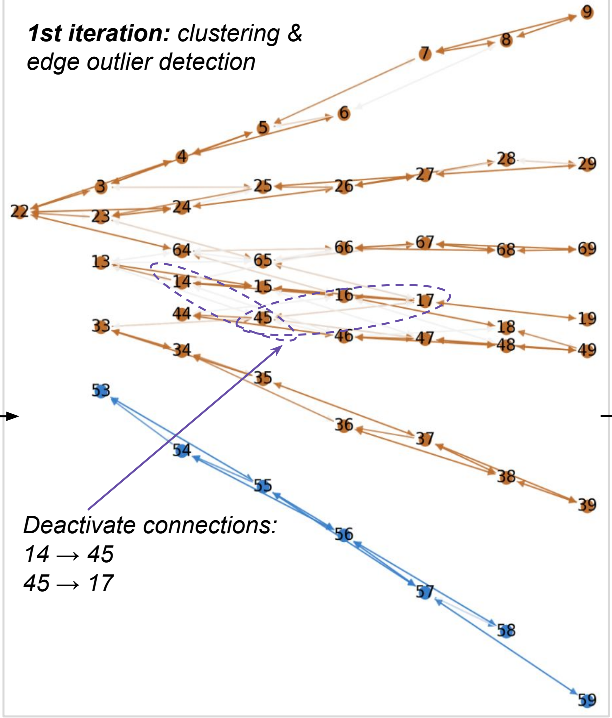
***Remaining network** after Iteration 1*



***Network Simulation**, weakly connected components*

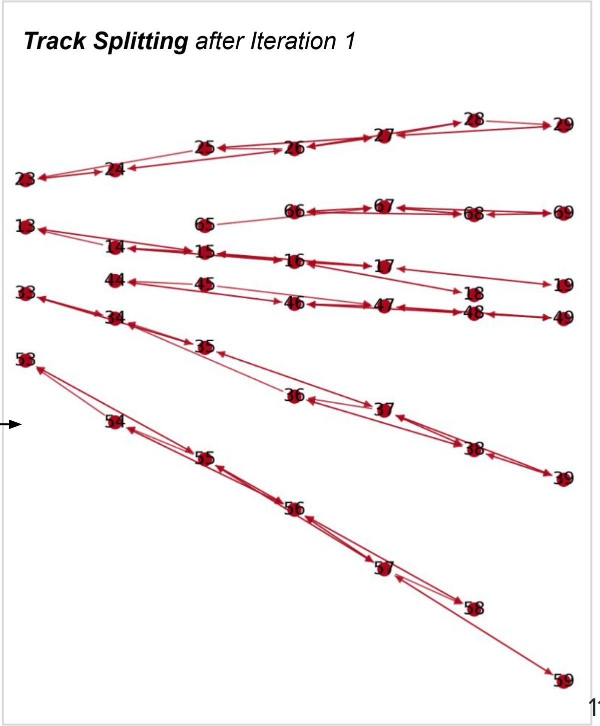


***1st iteration:** clustering & edge outlier detection*



Deactivate connections:
14 → 45
45 → 17

***Track Splitting** after Iteration 1*



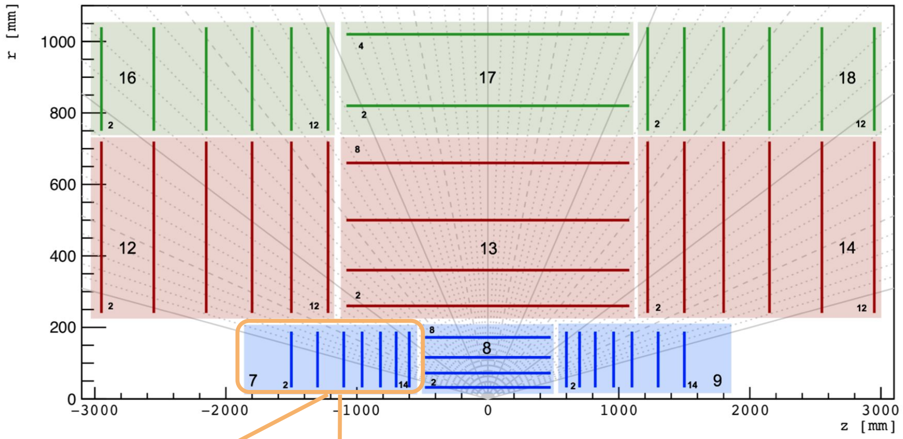
TrackML Model

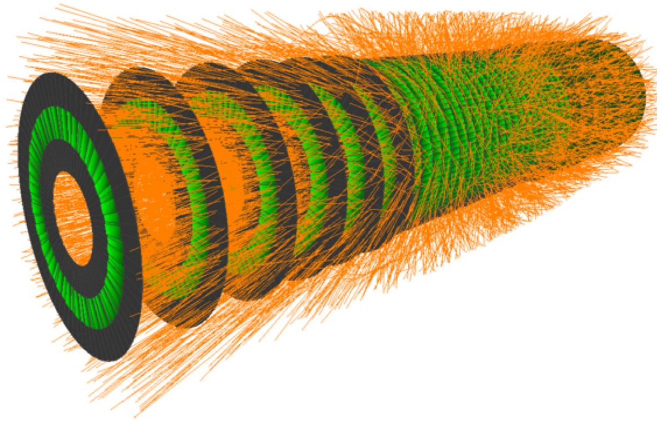
What is TrackML?

- More realistic detector model & generated data
- Open source (Kaggle) Data Science competition set out to improve the software needed to process & filter promising events

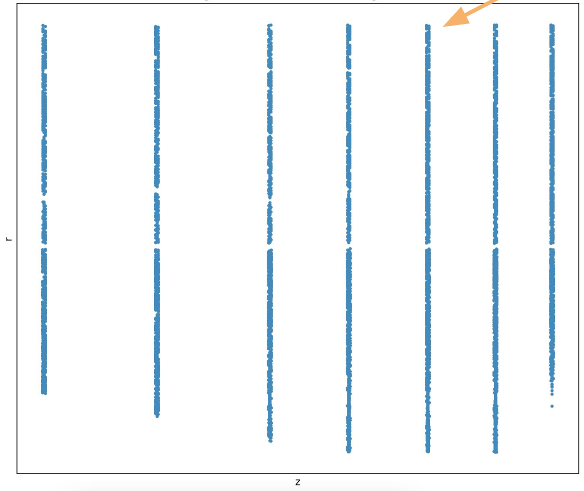
Using TrackML data:

- Here we focus on **Pixel Endcap volume (region 7 only)**
- An example of 1 event simulation
- Initialize network nodes & bidirectional edges
- **Number of nodes: 8748, number of edges: 12852**
- Nodes correspond to hits from a simulated event

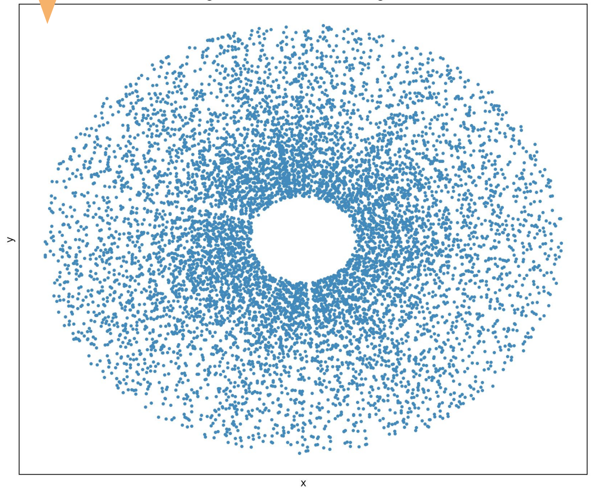




Nodes & Edges extracted from TrackML generated data



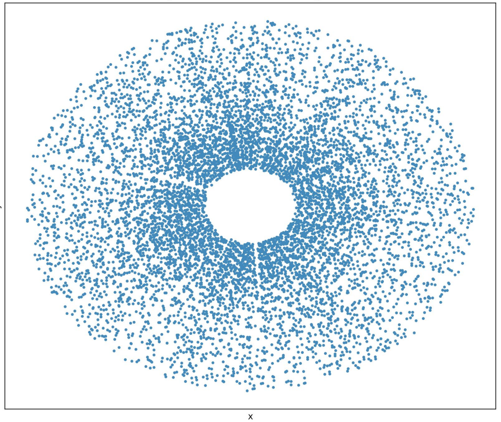
Nodes & Edges extracted from TrackML generated data



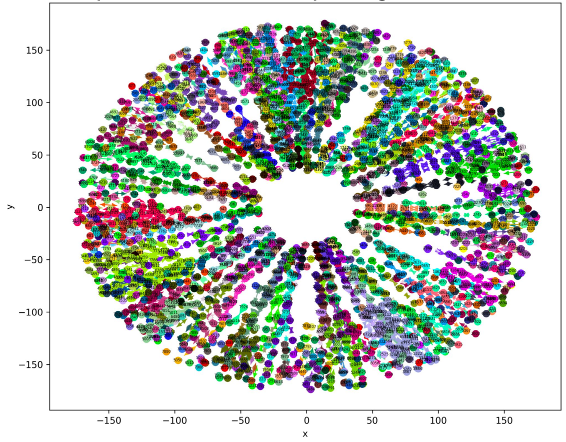
TrackML Results: Left Endcap

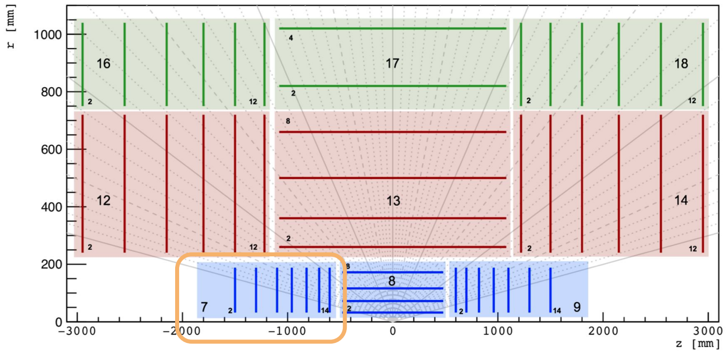
- Isolating the left endcap region
- 1683 subgraphs: 62% tracks extracted after iteration 1
- TrackML standard requirement: track purity & particle purity must be > 50%
- **Track reconstruction efficiency:**
- **> 90% efficiency for tracks with pT > 1GeV**

Endcap data points

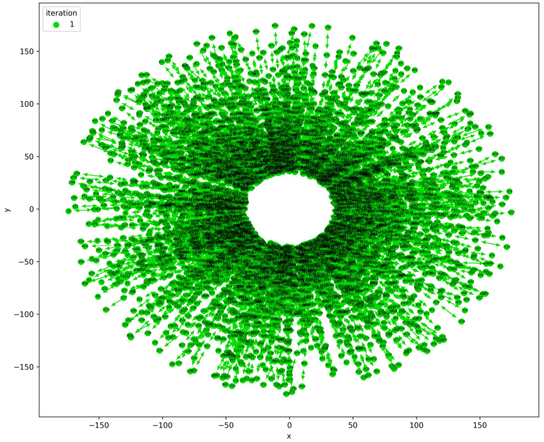


Graph network track splitting

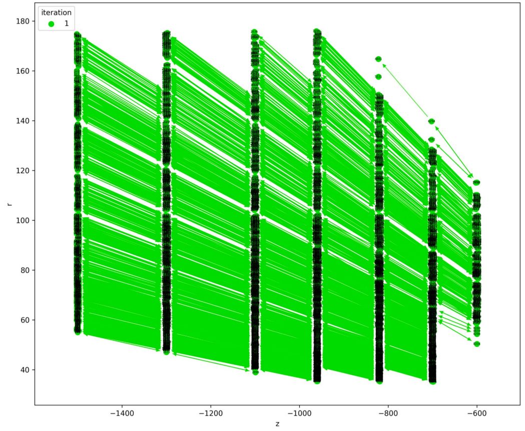




Extracted track candidates (xy)

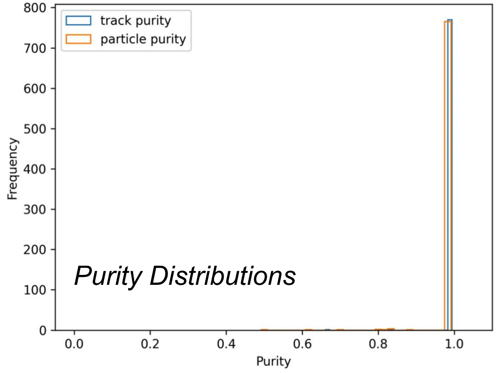


Extracted track candidates (rz)



62% track candidates extracted after 1st iteration uniformly

Track reconstruction efficiency: > 90%



Purity Distributions

TrackML Results: Pixel Barrel & Endcaps

***True Positive Rate (TPR):** the model correctly predicts the positive class, similarly for True Negative (TNR)*

- **Approximate timings → a couple minutes with no parallelization of code**
- Predicted class 1: Identifying an outlier edge
- Predicted class 0: Keeping an active edge on

Iteration 1:
- **Precision 96%, Recall 92%**
- Extracted candidates: 1629

Iteration 2:
- Precision 68%, Recall 48%
- Extracted candidates: 294

Iteration 3:
- Extracted candidates: 7

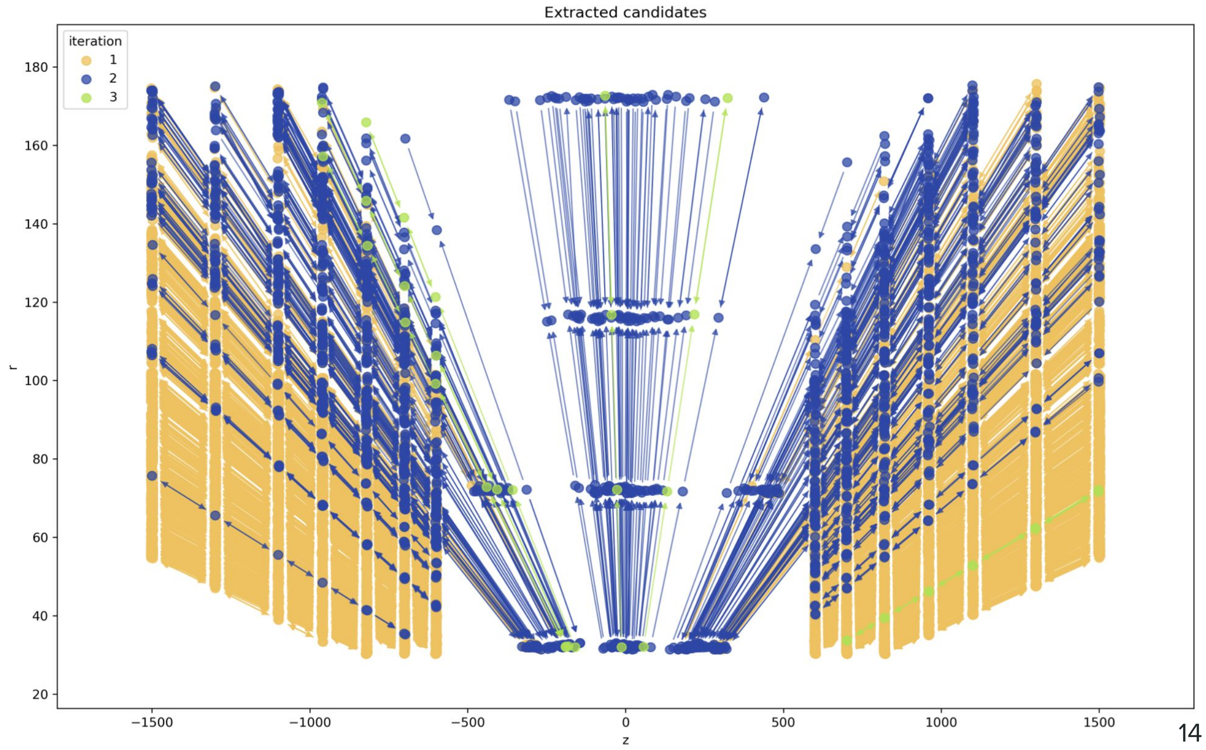
Confusion Matrices

Stage 1		Predicted Class	
		1	0
True Class	1	TP: 62102 (TPR: 0.92)	FN: 5211 (FNR: 0.08)
	0	FP: 2891 (FPR: 0.15)	TN: 16240 (TNR: 0.85)

Prediction of outlier edges

Prediction to keep edge active

Stage 2 Extrapolation		Predicted Class	
		1	0
True Class	1	TP: 18279 (TPR: 0.48)	FN: 19707 (FNR: 0.52)
	0	FP: 8786 (FPR: 0.54)	TN: 7528 (TNR: 0.46)



Summary

- The approach developed here is a **unique method**, unlike the traditional approach whereby Multi-Layered Perceptrons (MLPs) are trained

- The graph network is allowed to learn local track parameters on its own by **iteratively resolving ambiguities** in order to discover new track candidates

- Endcap: preliminary result of **> 90% track recon. efficiency** for tracks with **pT > 1GeV**

- Initial results with the barrel region show high purity of extracted tracks, **high TPR of 0.92 and TNR of 0.85**, at iteration 1

- The incorporation of **KFs for both information aggregation & track extraction is a unique approach** and has shown to give promising results on both a simple MC toy model & TrackML model

- **Neighbourhood information aggregation via message passing is a powerful feature** of GNNs which is leveraged here; the network has the ability to learn local track parameters

- Promising initial results on the Pixel barrel region. But further evaluation of this part of the TrackML detector is needed - further analysis will give a more complete picture in denser hit environment & give an indication of algorithm performance

Backup

The ATLAS Experiment and Inner Detector

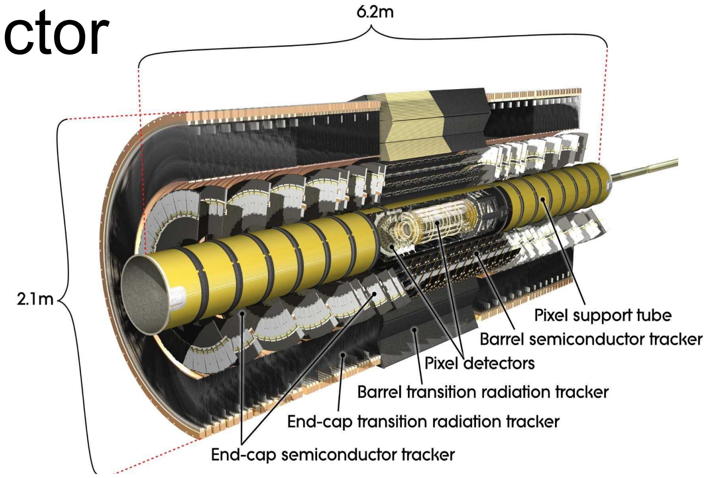
- General Purpose detector at the LHC, aims to make Standard Model precision measurements and test BSM theories

- The Inner Detector (ID) is dedicated to track and vertex reconstruction, it consists of 3 sub-detectors

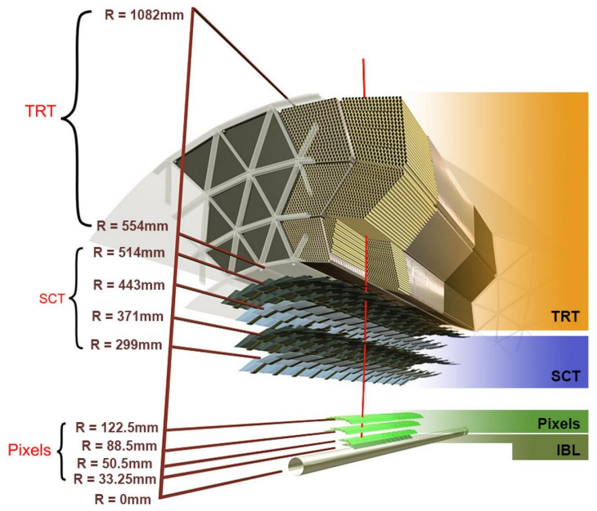
1. Pixel Detector and Insertable B-Layer (IBL);
   a. Closest to the beamline
   b. Has highest hit occupancy
2. Semiconductor Tracker (SCT)
3. Transition Radiation Tracker (TRT)

- The ID Trigger is part of the High Level Trigger (HLT) and performs fast online track and vertex finding.

- The ID Fast Tracking algorithm uses seeded track finding and combinatorial track following



6.2m
2.1m
Pixel support tube
Barrel semiconductor tracker
Pixel detectors
Barrel transition radiation tracker
End-cap transition radiation tracker
End-cap semiconductor tracker



TrackML Model Hit-Pair Predictor

Edge prediction for network nodes:

- We use the algorithm (FastTrack) submitted to the TrackML competition
- Github repo: https://github.com/demelian/fastrack

- Edge predictor algorithm:
    - In the Pixel detector there is a strong correlation between the lengths of pixel clusters and track inclination angle, found within data exploration
    - Train a binary classifier to identify hit-pairs which correctly belong to the same track
    - Discriminate between hit pairs that have correct hit association & hence belong to the same track, vs hit pairs that do not belong to the same track
    - Predicted results are converted into a to fast Look-Up-Table

Network Initialization

Given a node A, conditioned on its neighbourhood B_j:

- Compute X_{ij} (track state estimate)
- Compute C_{ij} (edge covariance)
- This forms a ***Gaussian component*** $\boldsymbol{\varphi_{ij}}$
- Store all components at node i
- ***Gaussian mixture*** $\boldsymbol{g_i(X)}$

$$\boxed{g_i(X) = \sum_j w_{ij}\varphi_{ij}(X, X_{ij}, C_{ij})}$$

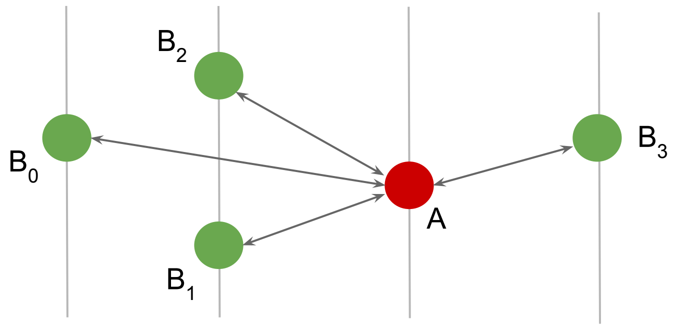
Definitions:

$\mathbf{e_{ij}}$
- **Bidirectional edge weight {0, 1}**
- **e = 0 inactive edge, e = 1 active edge**
- Arrow direction conventions: i → j transmission of messages from node i to node j

$\mathbf{w_{ij}}$
- **Mixture weight** for a Gaussian component transmitted from the neighbour node j to node i

$\mathbf{p_j}$
- **Prior probability** of neighbourhood nodes j and central node i being on the same track, if a track can produce at most one hit per layer

Initialize all ***bidirectional edges*** e_{ij} as “active” = 1



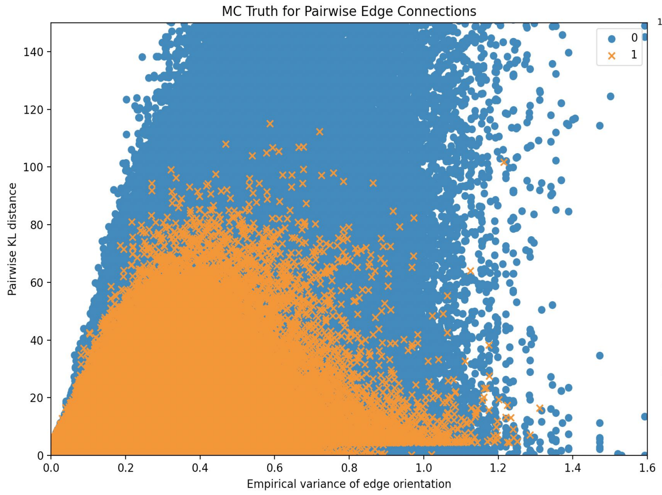
- **Priors:**
  p_1 & p_2 = 0.5, p_0 & p_3 = 1.0

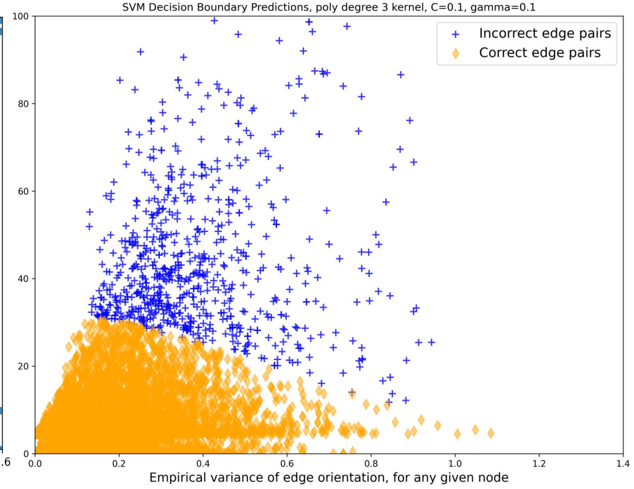
- **Mixture weight:**
  For each track state, initialize uniformly as 1/num nodes in neighbourhood of node A, w_{ij} = 0.25

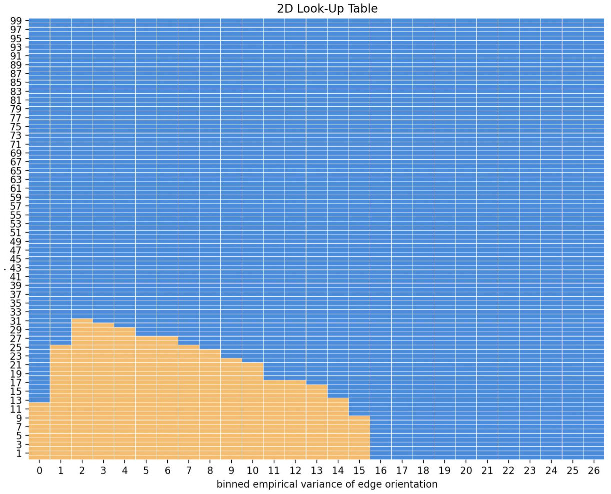
Learning the Optimal KL Threshold

- ***Can we obtain a functional form?***
    - Predict the optimal pairwise KL distance for correct edge pairs using MC truth
- ***Expectation:***
    - if the variance is high, tighter cut on the KL distance is needed (& vice versa)
- ***Simulation:***
    - 10000 events, each with 10 tracks, Sigma0 = 0.5
    - Focused on nodes in ‘cold’ regions
    - Train : test split 70 : 30
    - X feature vector: ***empirical variance of edge orientation***, for any given node & pairwise KL distance

- ***Training:***
    - *SVM trained to maximise recall*
    - *Optimum hyperparameters: Poly degree 3, C=0.1, ɣ=0.1*
- ***Predictions:***
    - *Recall (TPR) 0.94, tuned to 0.95*
    - *Decision boundary extracted & converted to 2D LUT for fast lookup*







Iteration 2: Information Aggregation & Kalman Filter

Definitions:

- Extrapolation:
  - Extrapolation of the track state estimate X, where F is the Jacobian of for the extrapolation from node B to A
  - Extrapolated covariance matrix C, with the addition of the process noise Q, and symmetrical product with Jacobian F
  - Jacobian F and process noise Q both derived using Ornstein-Uhlenbeck (OU model)
  - α determines the dynamics, it is a correlation hyperparameter
- Process Noise Q:
  - Q encompasses all material effects & is modelled by the OU process, which models the constant drift in ϕ due to the presence of the magnetic field
  - OU process noise models this dynamic behaviour
  - σ_{ou} is the uncertainty due to the OU process and σ_{ms} is the uncertainty due to multiple scattering
- Additional steps:
  - Residual r_{ij} with measurement at node A & extrapolation, where H is the measurement matrix
  - The covariance of the residual S_{ij}
  - The chi-squared distance $\Delta\chi_{ij}^2$

$$F = \begin{pmatrix} 1 & dx & g1 \\ 0 & 1 & f1 \\ 0 & 0 & e1 \end{pmatrix}$$

$$e_1 = e^{(-|dx|\alpha)}$$

$$f_1 = \frac{(1 - e_1)}{\alpha}$$

$$g_1 = \frac{|dx| - f_1}{\alpha}$$

$$Q = \begin{pmatrix} dx^2(\sigma_{ms}^2 + \frac{dx^2\sigma_{ou}^2}{4}) & dx(\sigma_{ms}^2 + \frac{dx^2\sigma_{ou}^2}{2}) & \frac{dx^2\sigma_{ou}^2}{2} \\ dx(\sigma_{ms}^2 + \frac{dx^2\sigma_{ou}^2}{2}) & \sigma_{ms}^2 + dx^2\sigma_{ou}^2 & dx\sigma_{ou}^2 \\ \frac{dx^2\sigma_{ou}^2}{2} & dx\sigma_{ou}^2 & \sigma_{ou}^2 \end{pmatrix}$$

$$H = [1 \quad 0 \quad 0]$$

$$dx = x_B - x_A$$

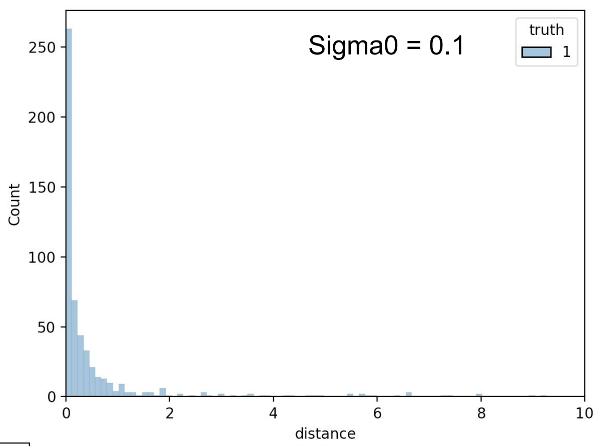
$$r_{ij} = m_A - H\tilde{X}_{ij}$$

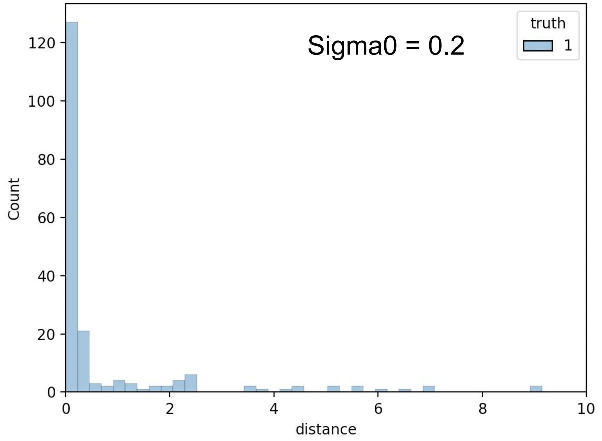
$$S_{ij} = H\tilde{C}_{ij}H^T + \sigma_0^2$$

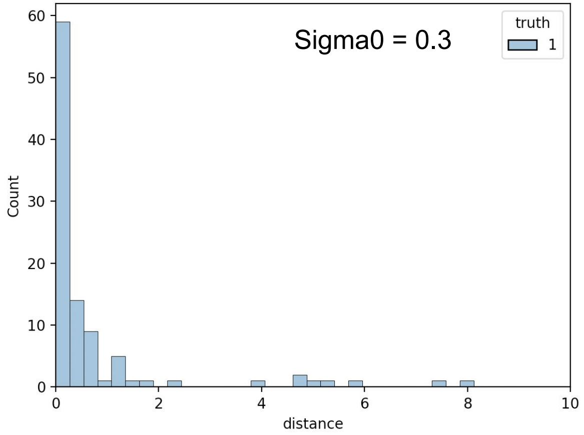
$$\Delta\chi_{ij}^2 = r_{ij}^T S_{ij}^{-1} r_{ij}$$

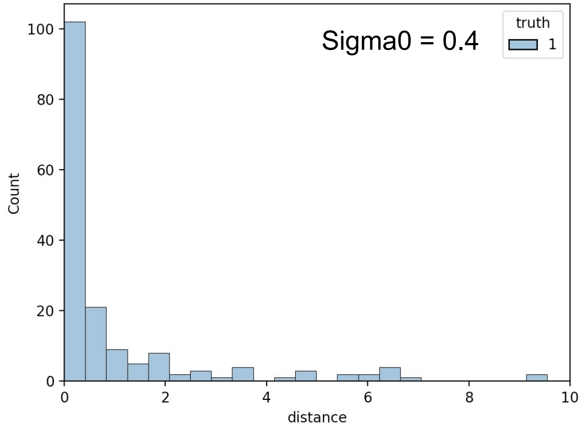
Chi2 Acceptance Threshold for Extrapolated Merged State

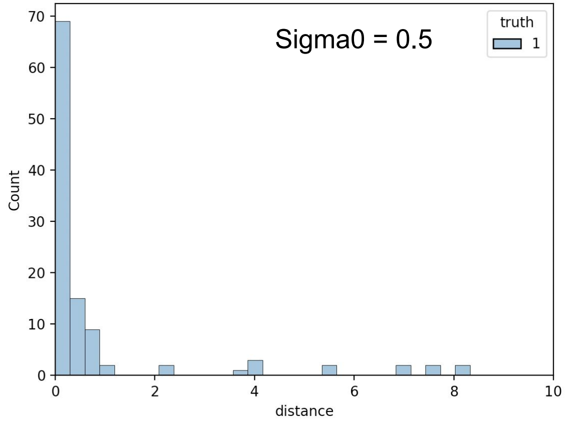
- Varying sigma0 and recording the chi2 distance for extrapolated states
- Plotted chi2 distance for states where the node and neighbour node has truth 1
- Distance is independent of sigma0
- Chi2 acceptance threshold set to 2





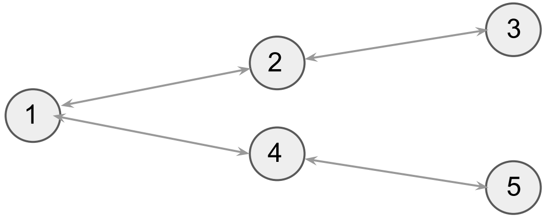


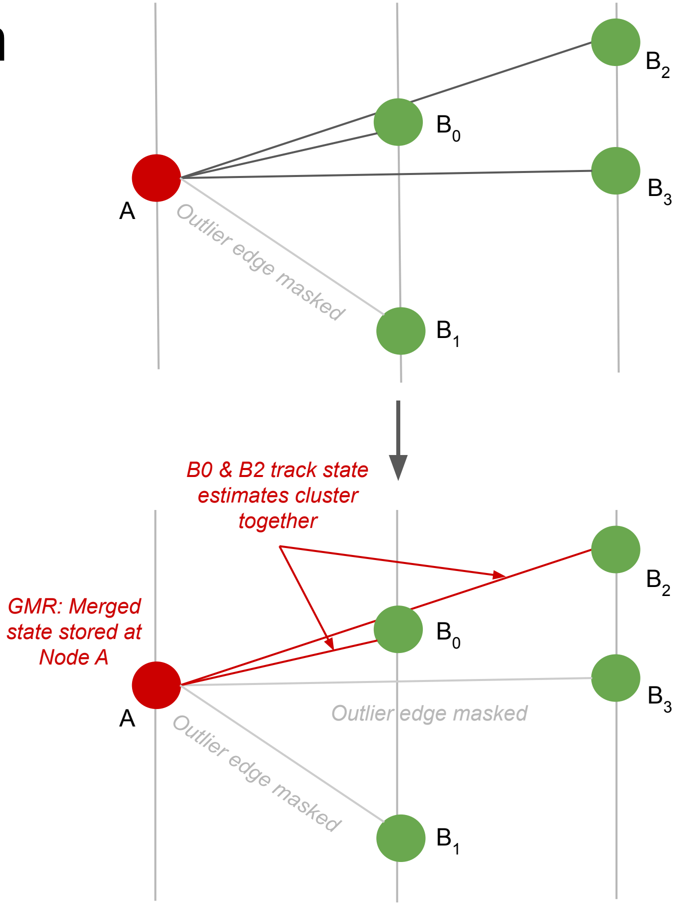




Iteration 3: Gaussian Mixture Reduction

- **Gaussian mixture reduction** is executed again on any **remaining networks** to further resolve ambiguities
- Perform **clusterization & merging on updated track states** outputted from the previous iteration
- Propagation of **higher precision estimates**
- Outliers are masked, priors & weights are updated
- Scenario where merging is forbidden:
    - A node has only 2 competing edge connections & components come from 2 different nodes in the same layer (i.e. nodes 2 & 4)
    - More precise extrapolated tracks from 2 & 4 will give sharper discrimination in later iterations




A
B_0
B_1
B_2
B_3
Outlier edge masked
B0 & B2 track state estimates cluster together
GMR: Merged state stored at Node A
Outlier edge masked
Outlier edge masked

Gaussian Components & Reweighting

Matrix definitions:

- At each node, all edge connections form a weighted Gaussian mixture
- Weights are given by w_ij and each edge connection is comprised of its track state vector X_ij and the edge covariance C_ij derived from the associated measurement errors

$$\boxed{g_i(X) = \sum_j w_{ij}\varphi_{ij}(X, X_{ij}, C_{ij})}$$

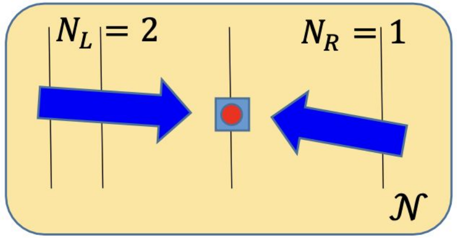
$$\boxed{\tilde{w}_{ij} = \frac{w_{ij}\beta_{ij}}{\sum_k w_{ik}\beta_{ik}} p_i}$$

- The edge weights are defined as a ratio, together with the prior probability p_i and the measurement likelihood B_ij defined as the normalized Gaussian measurement likelihood, as a function of the chi-squared distance between the extrapolated state and the measurement at the node

$$\boxed{\beta_{ij} = \left(2\pi|S_{ij}|\right)^{-1/2} e^{-\Delta\chi^2_{ij}/2}}$$

$$\boxed{\tilde{w}_{ij}/N_S}$$

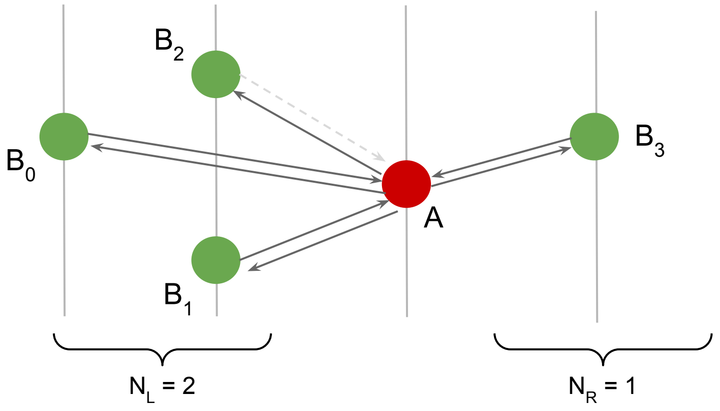
- The final weights are then divided through by the number of layers on each respective side of the MC toy model, i.e. N_L = no. of layers on left and N_R = no. of layers on right, as the track direction needs to be taken into account



Updating Priors & Weights

N_L = no. of layers on left
N_R = no. of layers on right

- As the network evolves edge connections will become masked/remain active → graph structure will change
- Hence reweighting mixture components & recomputing priors will be needed
- If any weights < threshold, these edge connections are isolated
- Forms part of the mechanism for edge activation/deactivation



Consider the example above:

- Connection B_2 to A is deactivated ($e_{B2A} = 0$)
- Message passing/extrapolation from B_2 to A is incompatible
- From the perspective of node A the priors change: p_0 remains at 0.5 (its previous value), p_1 gets updated 1.0 and p_3 = 1.0
- B0, B1 and B3 components will get reweighted

Reweighting the mixture:

- Measurement likelihood (conditional on component) given by β_{ij}

$$\boxed{\beta_{ij} = \left(2\pi|S_{ij}|\right)^{-1/2} e^{-\Delta\chi^2_{ij}/2}}$$

- Updated weights for the mixture extrapolated from node B_i are:

$$\boxed{\tilde{w}_{ij} = \frac{w_{ij}\beta_{ij}}{\sum_k w_{ik}\beta_{ik}} p_i}$$

- Account for the no. of layers in the neighbourhood
  - Need the probability that a track passing through node A was detected at layer L
  - Hence $\tilde{w}_{ij}$ must be divided by the no. of layers N_S on the corresponding side of the neighbourhood
  - I.e. B_0, B_1, B_2 division by 2, B_3 division by 1

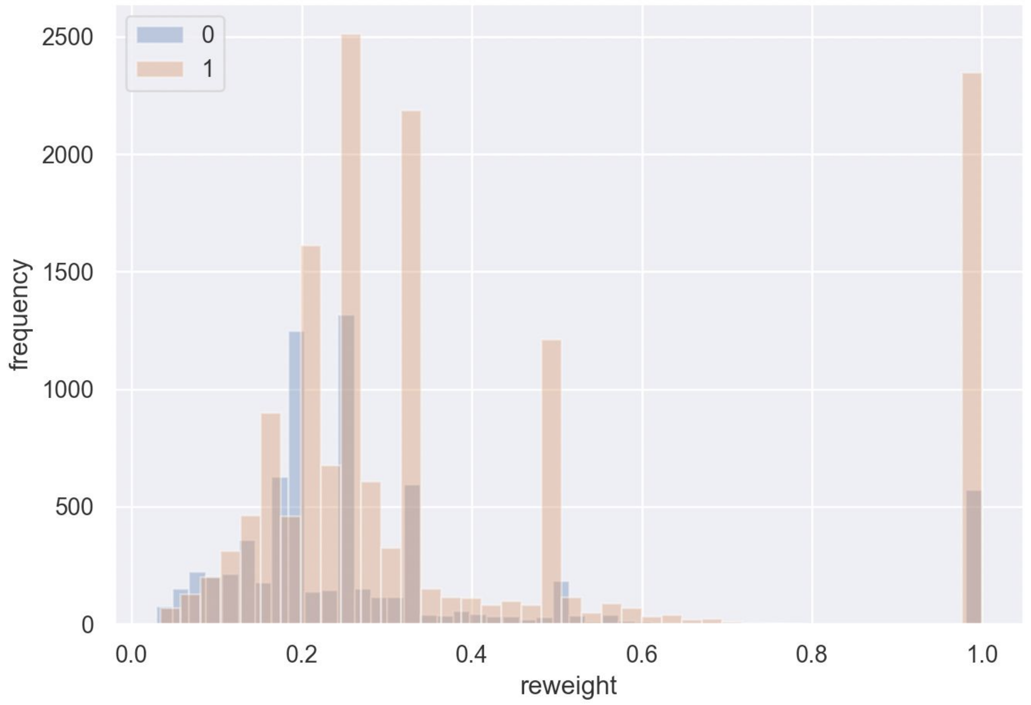
- Final Gaussian mixture with components:

$$\boxed{\varphi_{ij}\left(X, \tilde{X}_{ij}, \tilde{C}_{ij}\right)} \quad \boxed{\tilde{w}_{ij}/N_S}$$

Tuning the Reweight Threshold

- If the weights of components < threshold → deactivate these edge connections
- Plot: Edge connections reweight distributions:
- Tuning the reweight threshold parameter
- Look at remaining networks after the first reweighting calculation has been executed
- Edge MC truth (1 or 0) & edge reweight
- Both distributions overlap considerably, there is not a lot of difference separating the two distributions
- Therefore, as we do not want to turn off a large proportion of "good" edge connections, then reweight threshold of 0.1 is fine.

Reweight distribution after iteration 2 of edge connections for MC truth correct and incorrect edge pairs



Implementation of Kalman Filters

- KF is central to the tasks in this algorithm
- Kalman Filters implemented using Python package: **Filterpy**

Two main applications:

1. **Iterative Information Aggregation stage**
    - Connections/segments which we track as straight lines
    - Process noise term due to multiple scattering
    - Affects the track inclination variable
2. **Track Extraction**
    - Overall, the track bends due to the presence of the magnetic field
    - Need a more sophisticated process noise model in the KF for extraction
    - ***Ornstein-Uhlenbeck (OU) Process:***
        - Used as a type of correlated noise
        - Represents the constant drift in the track azimuthal angle (ϕ) caused by presence of B-field
    - KF model becomes 3-dimensional

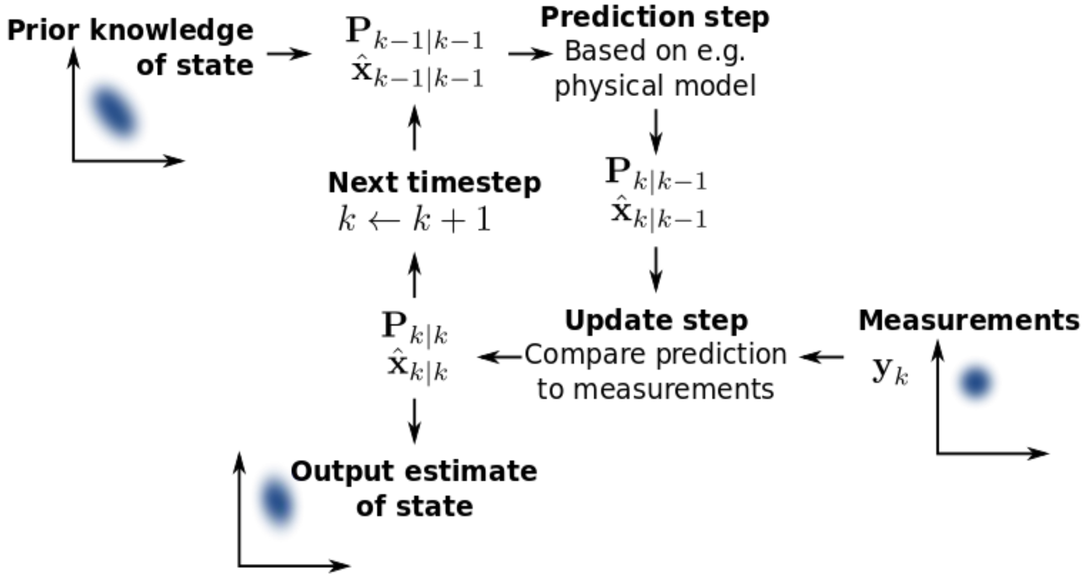
The KF Algorithm

Kalman filtering, also known as **linear quadratic estimation** (**LQE**), is an algorithm that uses a series of measurements observed over time, including statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.

Very powerful technique used in track fitting/trajectory optimization.

The algorithm works by a two-phase process. For the prediction phase, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with greater certainty.

The algorithm is recursive. It can operate in real time, using only the present input measurements and the state calculated previously and its uncertainty matrix; no additional past information is required.



Community Detection

- **Community structure of a network = division of its node set**
- Partition the network s.t. nodes in the same subgraph are closely connected & nodes in different subgraphs are sparsely connected
- Each subgraph is called a community

Aim:

- Resolve ambiguities earlier & aim for faster convergence.
- Extract communities within intersecting track candidates or >1 hit per layer

Modularity Maximization:

- Modularity: benefit function that measures the quality of a division of a network into communities, measures the relative density of edges inside communities w.r.t edges outside communities; high values correspond to good division
- Popular modularity maximization approach is the Louvain method (LM), iteratively optimizes local communities until global modularity can no longer be improved given perturbations to the current community state
- LM for directed networks: Integrates greedy strategy and hierarchical clustering, O(nlogn) where n is the number of nodes in the network

For a weighted, directed graph the functional form of modularity is given below:

$$Q_d = \frac{1}{m}\sum_{v,w}\left[\left(a_{vw} - \frac{k_v^{out}\cdot k_w^{in}}{m}\right)\cdot\delta_{C_v,C_w}\right]$$

Algorithm:

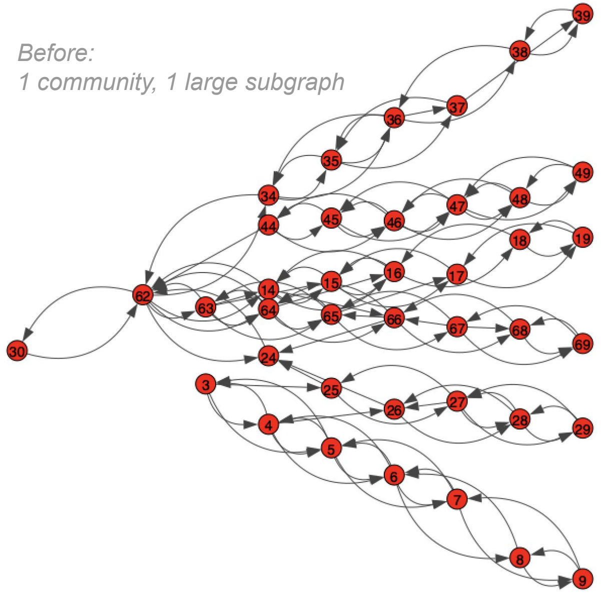
- Starting from singletons partition of vertices, the algorithm tries to increase the modularity by moving vertices from their community to any other neighbor one.
- Compute the gain obtained by removing vertex i from its own community Ci
- Repeat as long as it exists a move that improves the value of modularity

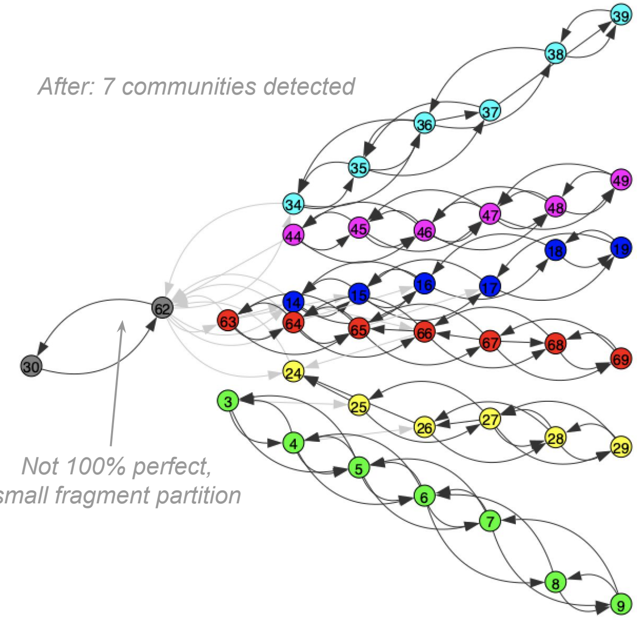
Community Detection

- **Community structure of a network = division of its node set**
- Partition the network s.t. nodes in the same subgraph are “closely connected”
- Distance metric: directed modularity
- Each subgraph/colour **represents a different community**
- Greater proportion of good track candidates established at an earlier iteration
- **Faster convergence**

Track Splitting:

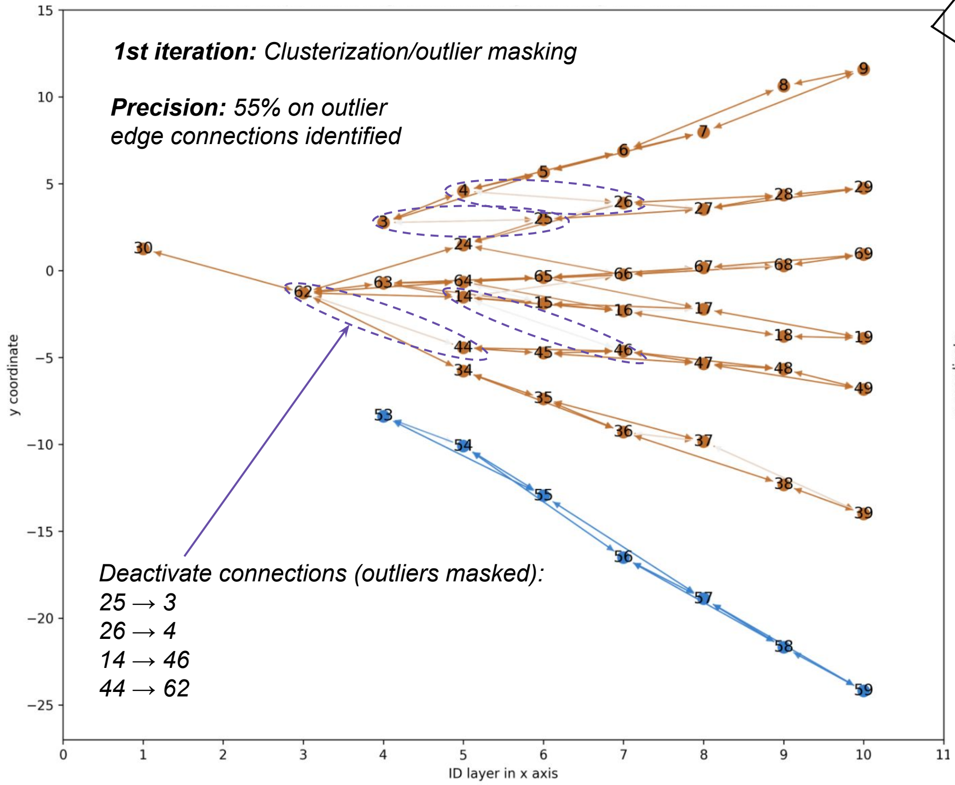
$$\begin{tikzcd}\text{1. Connected Component Analysis} \arrow[r] & \text{2. Kalman Filter Track Fit} \arrow[r] & \textbf{3. Community Detection}\end{tikzcd}$$





Results: Example 2

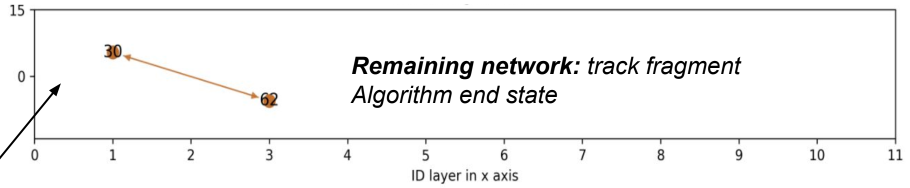
Iteration 1 only



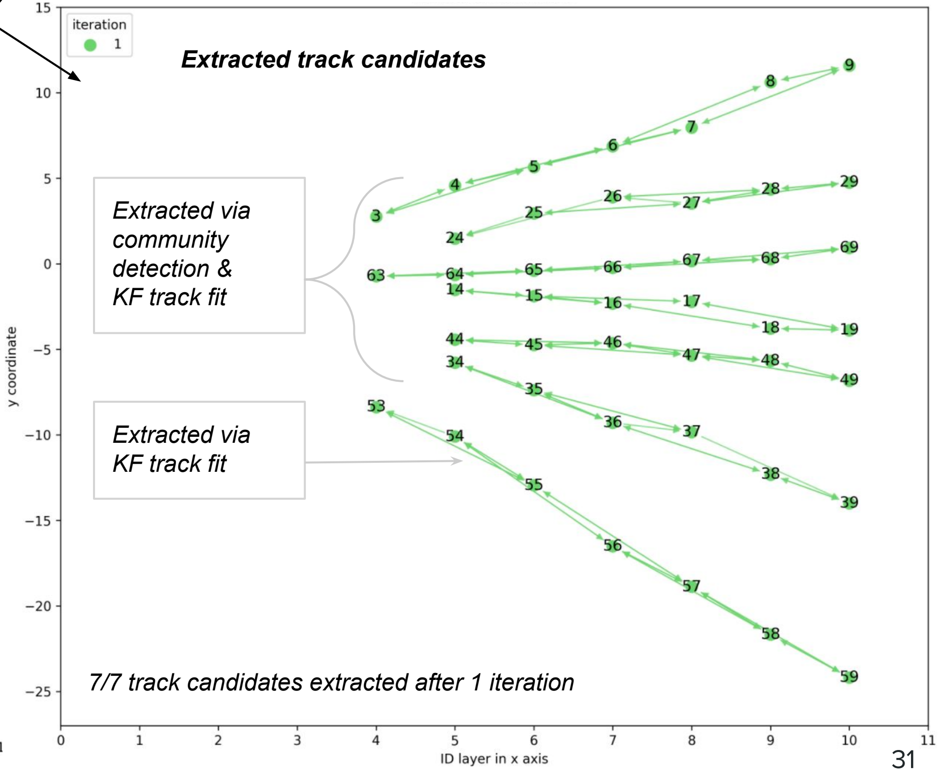
1st iteration: *Clusterization/outlier masking*

Precision: *55% on outlier edge connections identified*

Deactivate connections (outliers masked):
25 → 3
26 → 4
14 → 46
44 → 62



Remaining network: *track fragment*
Algorithm end state



Extracted track candidates

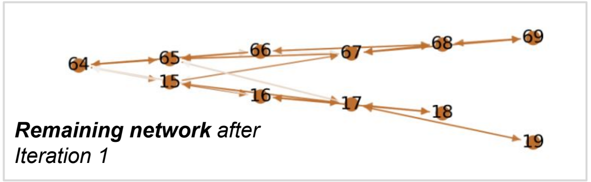
Extracted via community detection & KF track fit

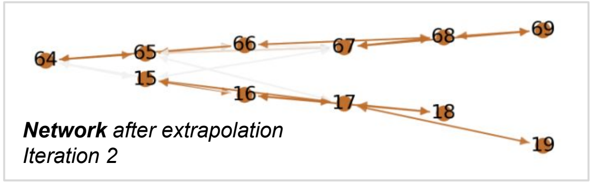
Extracted via KF track fit

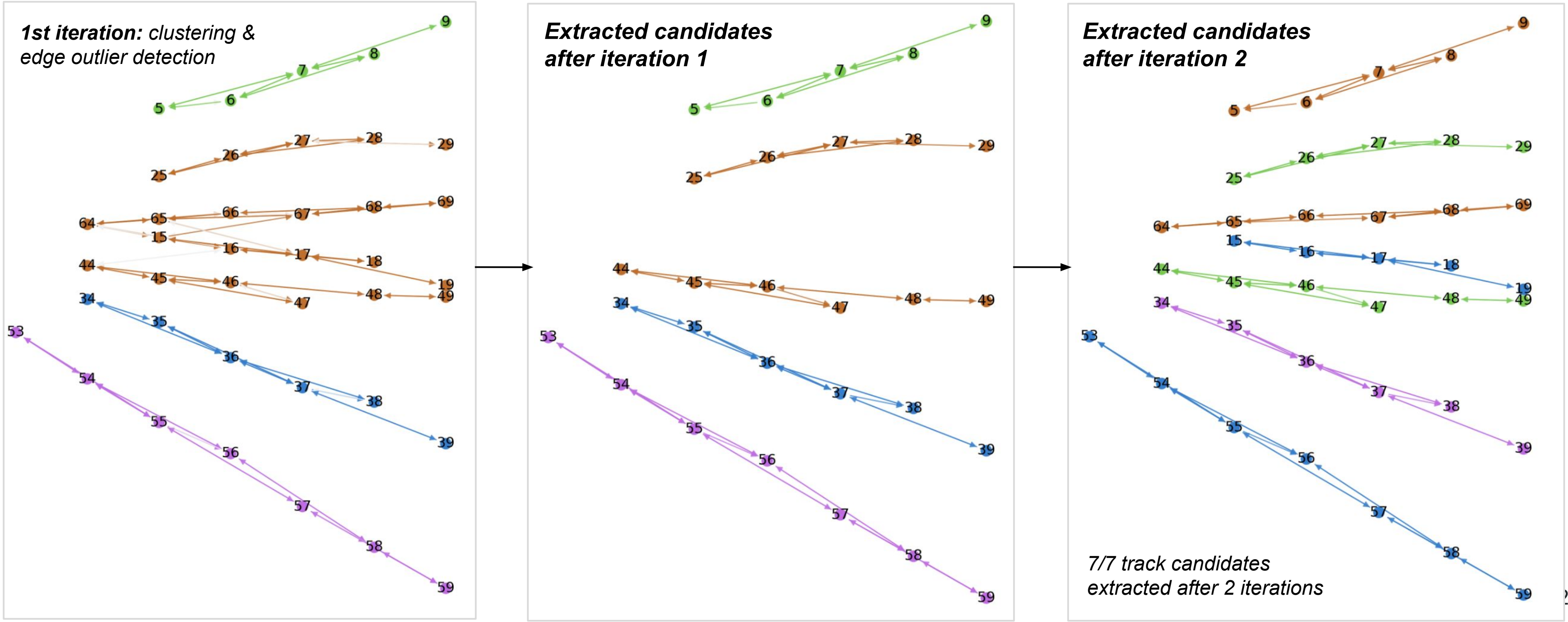
7/7 track candidates extracted after 1 iteration

Results: Example 3

- Iteration 1 & 2
- *This specific simulation was able to resolve all ambiguities within 2 iterations*
- *100% purity wrt MC truth*


Remaining network *after Iteration 1*


Network *after extrapolation Iteration 2*


1st iteration: *clustering & edge outlier detection*

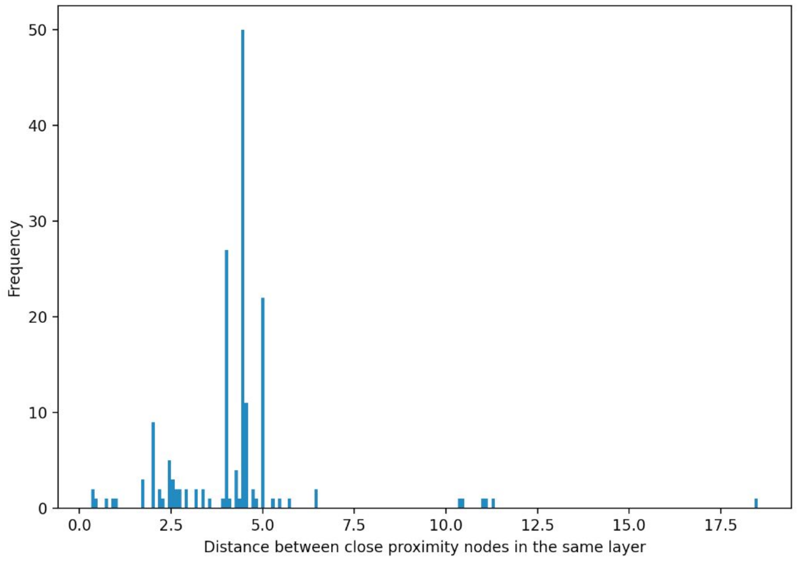
Extracted candidates after iteration 1

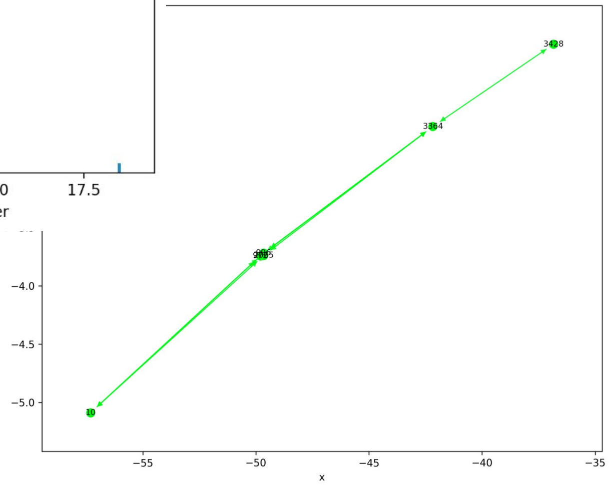
Extracted candidates after iteration 2

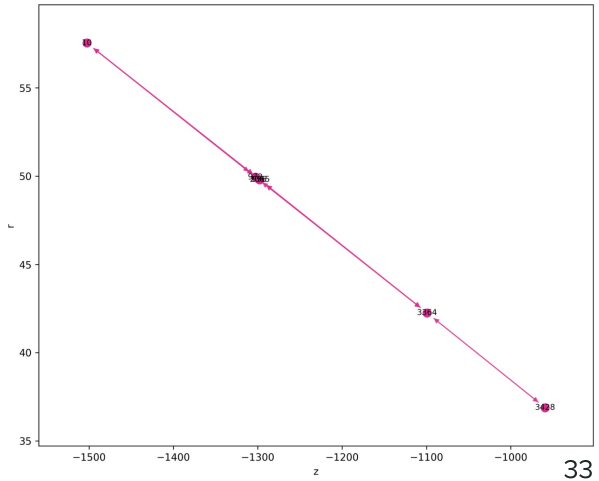
7/7 track candidates extracted after 2 iterations

TrackML - Node Merging

- Tuning of 3d separation distance - first considered module_id
- However not easy to calculate how close each hits are to each other due to arrangement of modules
- Calculation of 3d separation of the 2 nodes → threshold from distribution
- Small improvement - stats on the endcap: 515/1683 → 583/1683 subgraphs extracted on entire endcap region in iteration 1
- Will be useful for first iteration as these track candidates have been extracted first, and will make further iterations much faster
- Flexible in the sense that at the moment I am only looking for a specific type of subgraph (Any number of layers containing 2 nodes → apply distance cut)







Track Purity, Particle Purity, Track Reconstruction Efficiency

- Only particles with four hits or more are considered, and only proposed tracks with four hits or more are considered
- Each track is matched with the ground truth ***majority particle*** sharing with it the greatest hit number

- ***Track purity:***
  - The ratio of this intersection to the number of hits of the reconstructed track
- ***Particle purity***:
  - The ratio of this intersection to the number of hits of the underlying particle

- Both ratios > 50% to define a good track so that a one-to-one relationship between particle and track can be defined
- Same definition as what is used in the TrackML study

Track Reconstruction efficiency:

- We consider reference particles which are fully contained in the volume we are currently working on (endcap volume 7), this defines our reference tracks

$$Reconstruction\ efficiency = \frac{No.of\ successfully\ reconstructed\ tracks}{No.of\ reference\ tracks\ in\ volume\ of\ interest}$$

Ongoing & Next Steps

- *Investigating problems in the* ***Barrel layers***
  - Analysing the properties of the remaining network connections in the Barrel
  - Barrel - to - Endcap transition connections
- ***Improving the purity of results:***
  - Confusion matrix for iteration 2 can be significantly improved
  - Extrapolation - in both (x, y) and (r, z) planes
- ***Community Detection***
  - Implementing a custom community detection
  - Analysing node pairs & utilizing KL distance to identify communities
- *Implement* ***stopping criteria***
  - I.e. Quality of remaining networks < threshold, terminate the algorithm
- ***Performance Studies/Testing***
  - Applying the algorithms to many events and averaging statistics
  - Optimizations - improve implementation i.e. parallelize code