Imperial College
London

23 March 2022

Fast Primary Vertex and Track Reconstruction Methods

SwiftHEP workshop

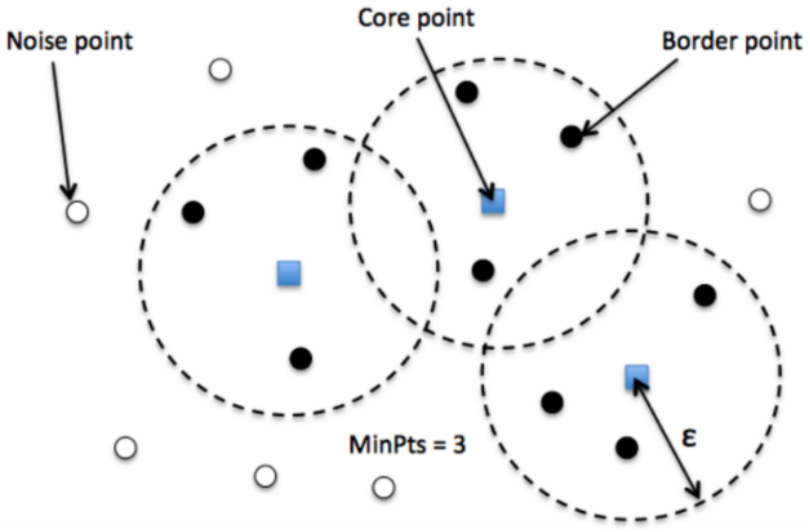
Lucas Santiago Borgna

Imperial College London

Table of Contents

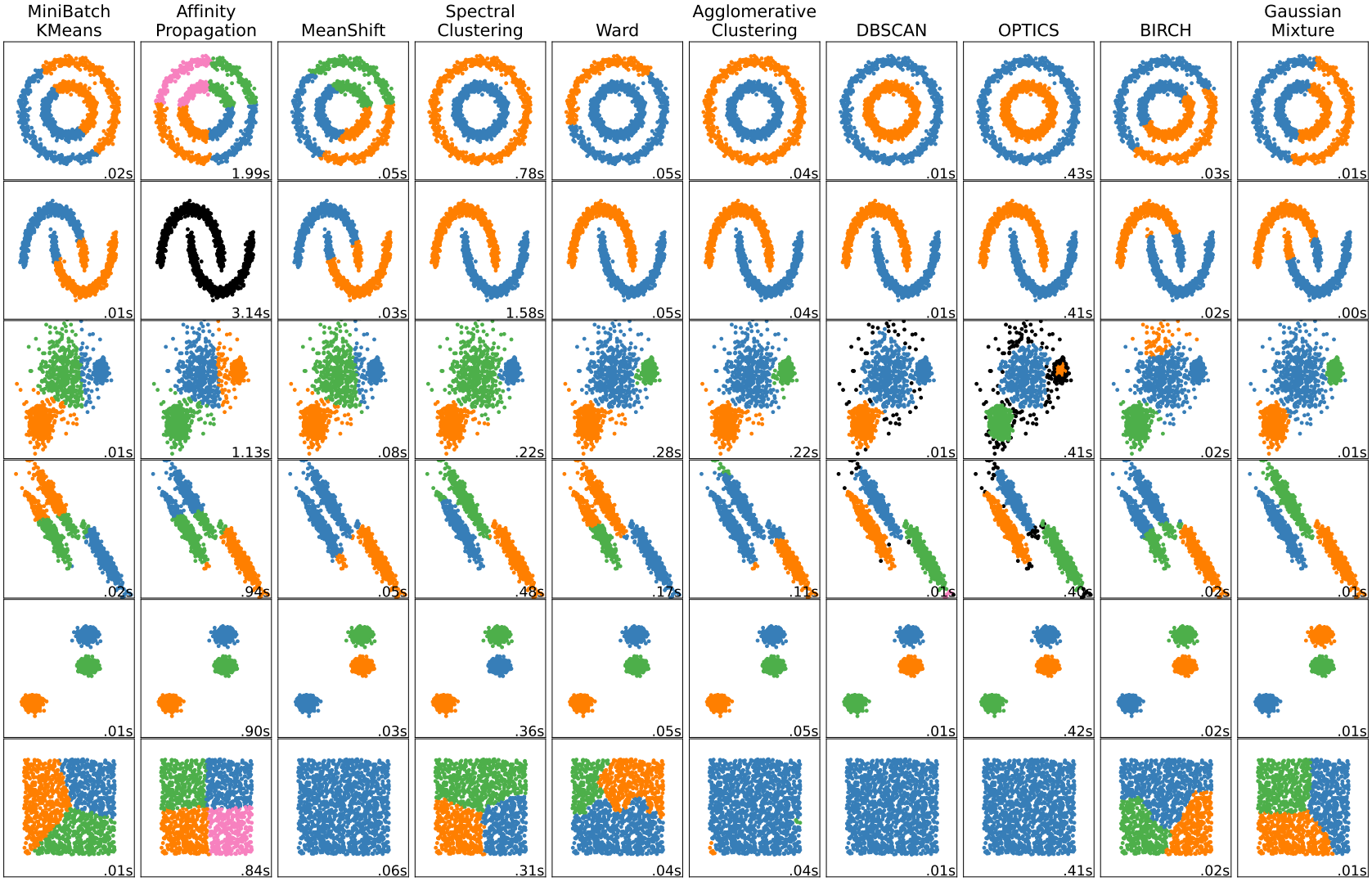
Primary vertex reconstruction and DBSCAN

DBSCAN = Density Based Clustering of Applications with Noise



DBSCAN paper

- DBSCAN is an unseeded/ unsupervised clustering algorithm, with 2 main hyperparameters
- **minPts** = minimum number of points required to form a cluster.
- ε =**eps** = maximum distance between two points for one to be considered a neighbor of the other.



example reference code

DBSCAN Applications

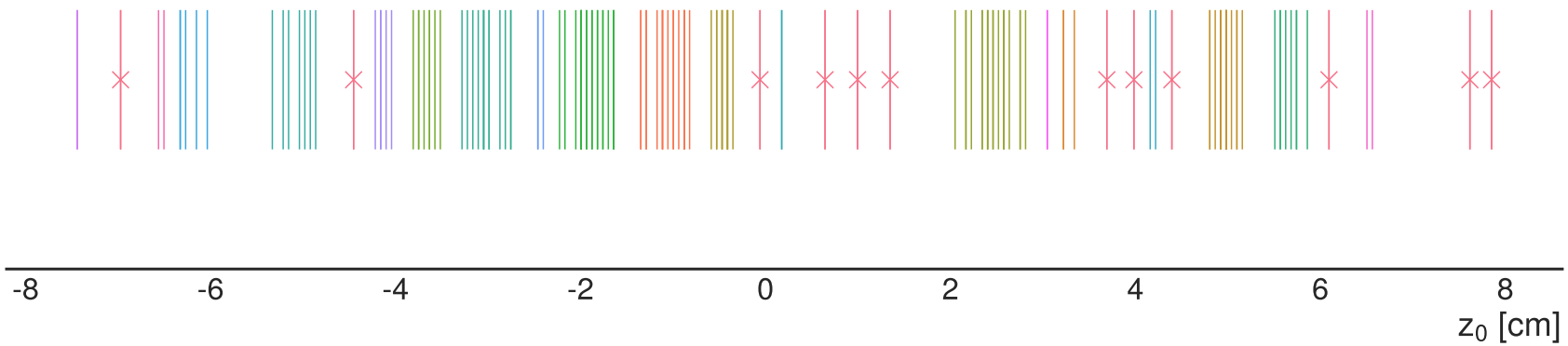
DBSCAN has a wide range of applications.

- Primary vertexing is a simple 1D application of DBSCAN as a proof of principle.
  - Challenging to port to FPGA due to sorting and iterative nature.
  - Helpful to gain experience with the algorithm.
- DBSCAN can also be used for track finding with GNNs, as shown in this paper.
- Clustering in the HGCAL upgrade can also be done with DBSCAN.

Imperial College London

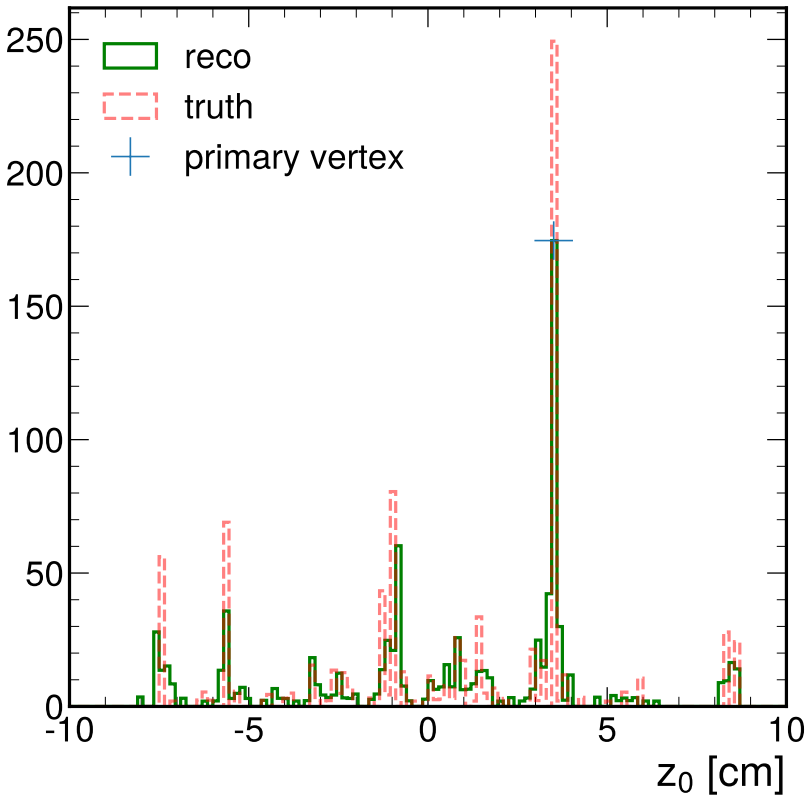
Primary vertexing - a test case

- CMS Phase-II, can use L1 Tracks for primary vertexing.
- Primary vertexing helps the trigger decision.
- Improves event reconstruction or pile-up suppresion.
- Based on track z_0 information.
- L1 trigger takes 4 μs to make trigger decision
- up to a maximum of 1665 tracks to read.



Imperial College London

FastHisto



- FastHisto is the current benchmark.
- Histogram the track z_0 and weight them by p_T.
- Highest peak corresponds to the primary vertex.
- only hyperparameter is the width of the bins.
- Computational complexity $\mathcal{O}(n_{bins} + n_{tracks})$

FastHisto and DBSCAN comparison

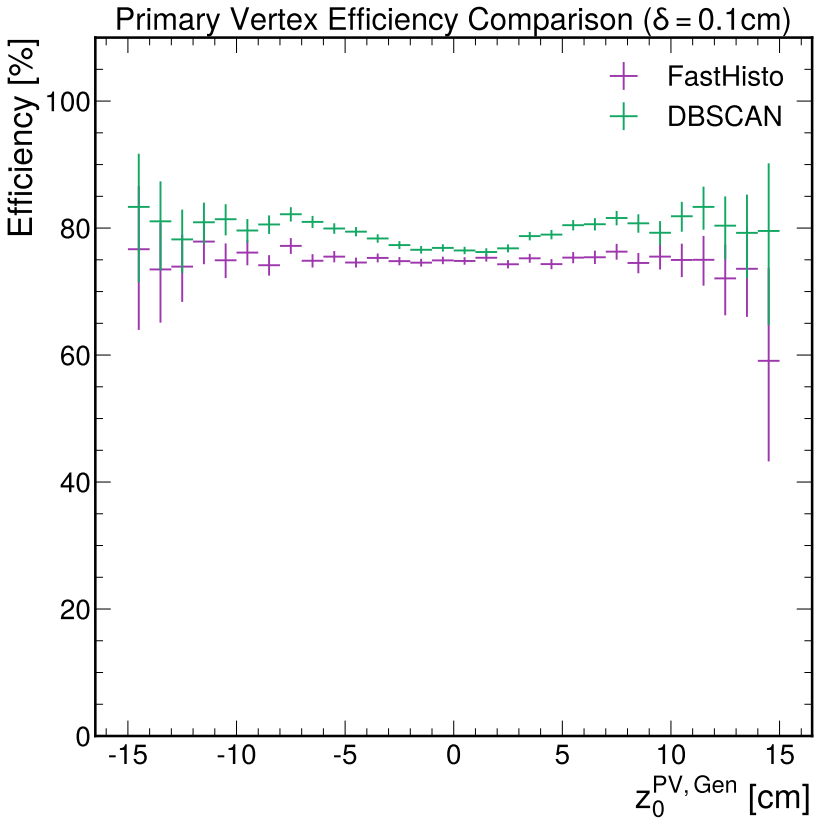
Comparison between FastHisto and DBSCAN using CMS Phase-II $t\bar{t}$ data.

- FastHisto uses 256 equaly spaced bins from -15 to 15 cm
- DBSCAN uses **minPts=2** and **eps=0.08 cm** for optimal configuration.
- comparison is done offline, using python and sklearn packages for DBSCAN.
- Helps to build experience and sanity check the performance of the algorithm.

Previous in-depth comparison shown in this thesis.

Imperial College London

Primary vertex efficiency

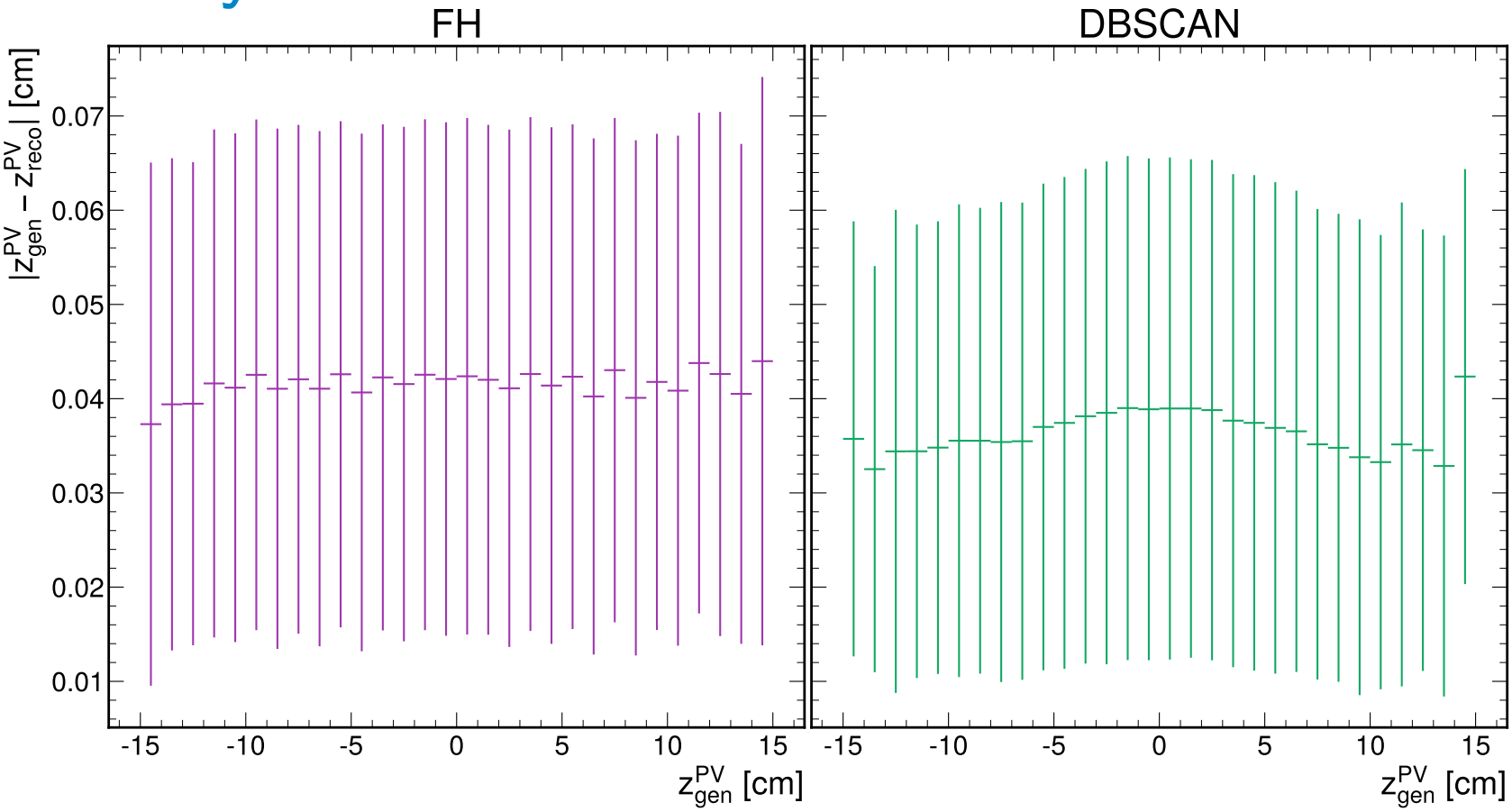


Comparison of the primary vertex reconstruction efficiency as a function of $z_0^{PV,Gen}$.

- DBSCAN average efficiency 78.2 %
- FastHisto average efficiency 75.0 %

Imperial College London

Primary vertex resolution



DBSCAN has a consistantly better resolution throughout $z_0^{PV,Gen}$.

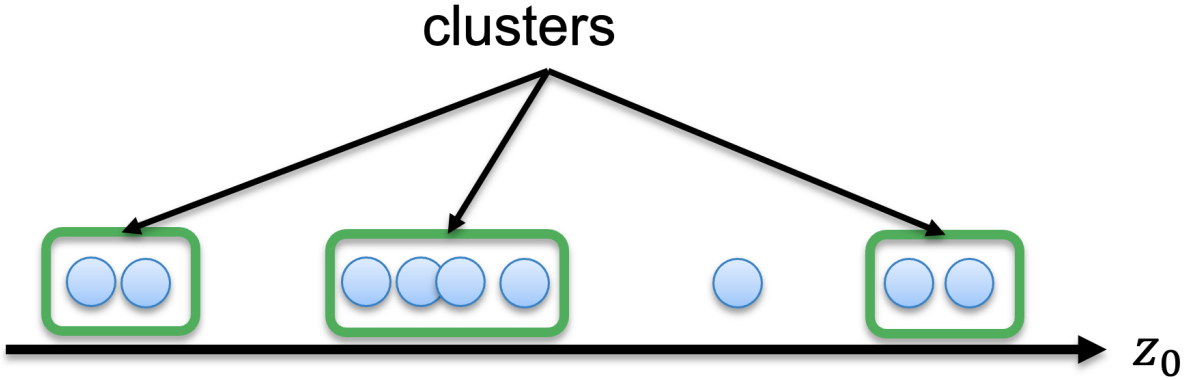
Track to vertex association

- Correctly identifying tracks as belonging to the primary vertex, **track-to-vertex association**, binary classification problem
- FastHisto and DBSCAN perform very similarly to each other, FastHisto marginally better.

Metric	**FastHisto**	**DBSCAN**
TPR	0.927	0.926
FPR	0.143	0.149
AUC	0.892	0.889

Overall, the performance of DBSCAN seems sensible

Accelerated DBSCAN



isLeftBoundary?	1 0	1 0 0 0	1	1 0
is Boundary ?	1 1	1 0 0 1	0	1 1
Boundary Indices	0 1	2 ∞ ∞ 5	∞	7 8

Boundary Indices sorted array = $[0, 1, 2, 5, 7, 8, \infty, \infty, \infty]$

Clusters boundaries = $[0, 1], [2, 5], [7, 8]$

More details can be found in appendix.

Accelerated DBSCAN implementation

Resource usage of full DBSCAN implementation

- latency 0.726 μs, 100 MHz clock, VU9P, Maxeler DFE.
- CPU latency 92.7 μs, single-threaded, C++, 127$\times$ speedup.
- Bitonic sort resposible for a large portion of resource usage

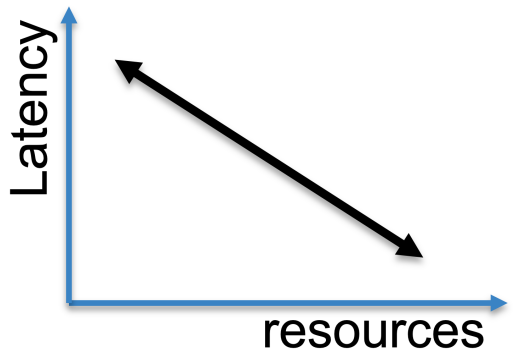
Logic	Use	Total	Percentage [%]	Bitonic sort only percentage [%]
Logic utilization	1725099	3546720	48.6	11.2
LUTs	422811	1182240	35.8	7.3
FF	1302288	2364480	55.1	13.2
DSP	0	6840	0	0
BRAM18	611	4320	14.1	8.3
URAM	118	960	12.3	8.1

Accelerated DBSCAN results

- Resource usage is very high.
- sorting networks are a large part of the resource usage.
- firmware is capped at a maximum input of 232 tracks, needs to be able to handle up to 1665.
- Currently, sorting the 1665 tracks would use 80.4 % of the logic and 94.7 % of FF.
- firmware runs all input tracks in parallel.

Accelerated DBSCAN Further work

- Minimize resource usage.
- Parametrize the algorithm by the number of input tracks.
- Investigate the parametrized latency



DBSCAN summary

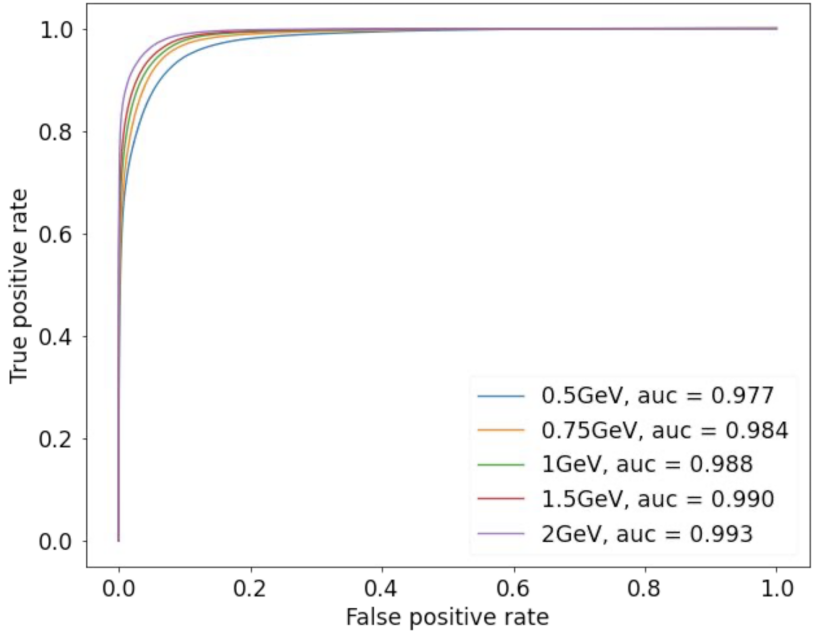
- Compared the physics benchmarks between FastHisto and DBSCAN offline.
- DBSCAN shows improved primary vertex efficiency and resolution.
- Similar track-to-vertex classification performance.
- Accelerated DBSCAN can run in 726 ns
- High resource usage, mostly due to Bitonic sort.
- Working to scale up the algorithm.

Reinforcement learning for track reconstruction - Liv Våge

Graph neural net for tracking

For an in-depth descriptions, see Liv's previous SwiftHEP talk here

- A track can be represented as edges between hits.
- Graph neural nets can predict edges between nodes to reconstruct tracks.
- This is perhaps the most promising ML for tracking approach.

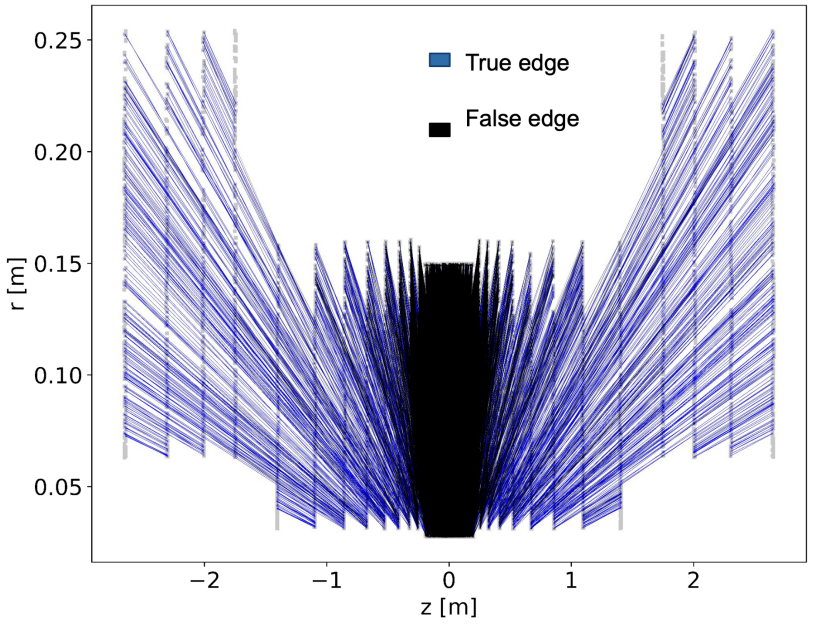


Edge classification accuravy on CMS Monte Carlo data benchmarked at various minimum p_T marks.

Limitations of graph neural nets for tracking

The work follows this paper, but makes use of CMS MC data.

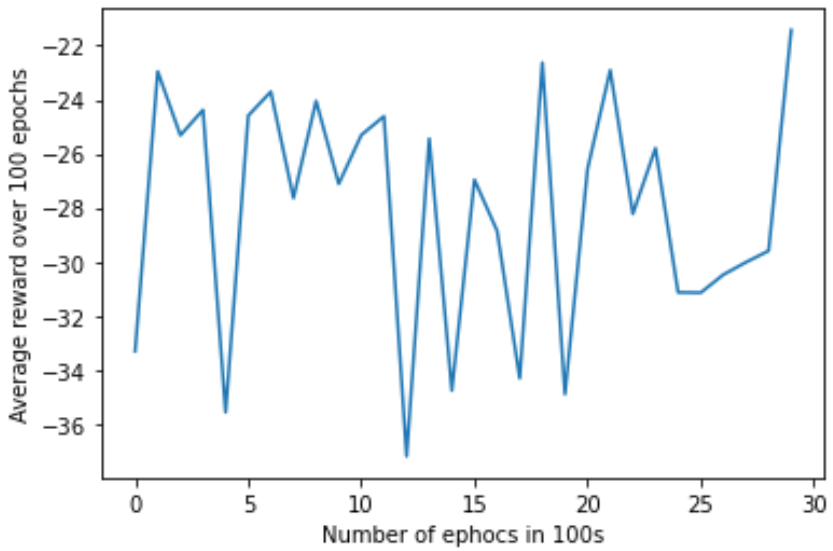
- Building graphs is slow.
- Current work doesn't allow for missing hits and removes multiple hits in a layer for a given particle.
- Most work uses only the inner part of the detector an p_T thresholds because of the large graph size.



Built graph for one event, $p_T > 0.5$ GeV.

Reinforcement learning for graph building

- Reinforcement learning could contribute to building smaller graphs.
- This would also remove restrictions like no missing hits.
- DDPG and other reinforcement learning methods are currently being explored.
- The agent in reinforced learning is very parallelisable, making it a good candidate for acceleration.



Reward is distance from true hit

HyperTrack - Mikael Mieskolainen

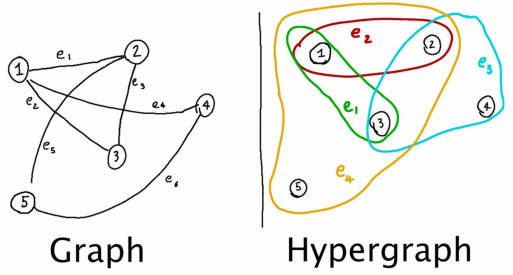
Imperial College London

HyperTrack: Deep Learned Graph Clustering

'**Pure AI**' approach, somewhere between supervised--unsupervised--reinforcement learning. Solving a hard combinatorial problem with a variable input-output set cardinality (size). The problem cluster structure and clustering sequence is learned, not enforced by hand or by any fixed metric.

Highly generic approach: based on graph nets + transformers. Realistic online tracking will be probably more efficient with helix driven heuristics – Equations of Motions are simple and efficient → a hybrid classic-AI method probably the winning solution. Obvious target here are physics analyses with hard final state combinatorics involved or very hard clustering.

Next: Crucial further development may be needed in Monte Carlo Tree Search like expansions in the intermediate steps (c.f. DeepMind AlphaGo/Zero) → *unsolved* in general how to make such algorithms faster.



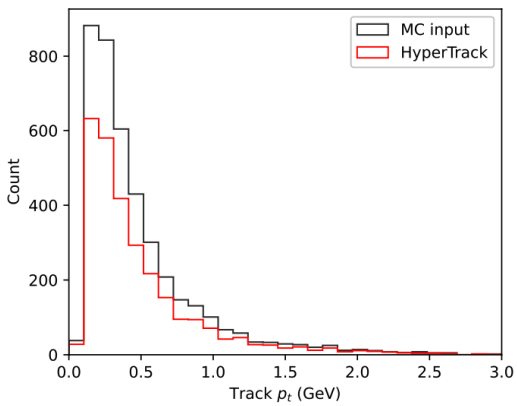
Full model
[kNN] + [GNN & edge MLP] + [Greedy diffusion / MC Tree Search] + [Transformer & MLP] + [GNN & node MLP]

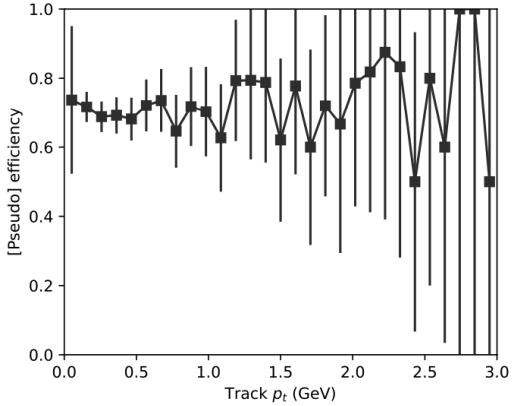
Info: m.mieskolainen@imperial.ac.uk, github.com/mieskolainen

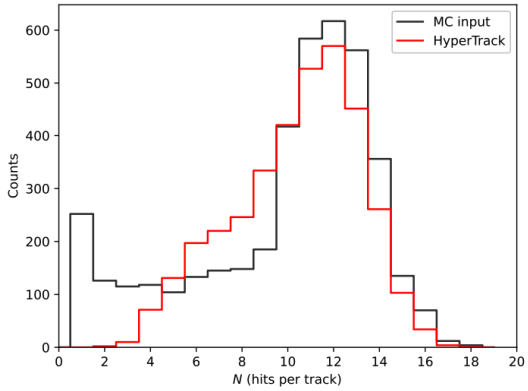
Imperial College London

Small scale tests

Using TrackML Kaggle challenge MC data and small 8 GB GPU







Preliminary!

Proof-of-principle: Near end-to-end autograd trainable implementation for a track reconstruction. Interesting results but large scale utilization may require conceptual modifications from sequential clustering to parallel operation, to be able to load GPU maximally etc.
[pytorch based], ML references: Amortized clustering, Set transformers

Summary

Summary

- Shown the potential performance of DBSCAN and referenced it to the current FastHisto benchmark.
- Accelerated DBSCAN in firmware using maxeler DFE.
  - Low latency 726 ns, high resource usage.
- Reinforcement learning can help to speed up the graph building part of GNN for tracking.
- Proof-of-principle HyperTrack for tracking.

Appendix

Accelerated DBSCAN - algorithm summary

1. sort tracks by z_0
2. prefix sum p_T
3. find left boundaries
4. find right boundaries
5. create boundary indices
6. sort boundary indices
7. calculate vertices and sum of p_T
8. sort vertices by p_T