Accelerated graph building

for particle tracking graph neural nets

Lucas Borgna, Alex Tapper, Liv Våge

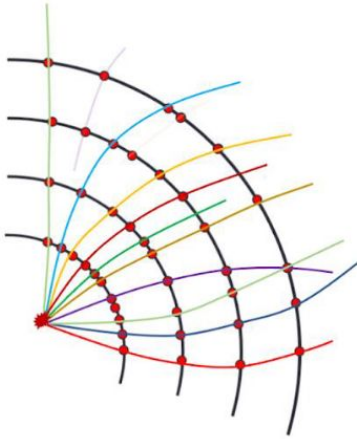
Connecting the Dots 31.05.2022

Imperial College London

liv.helen.vage@cern.ch

Link to slides with animation

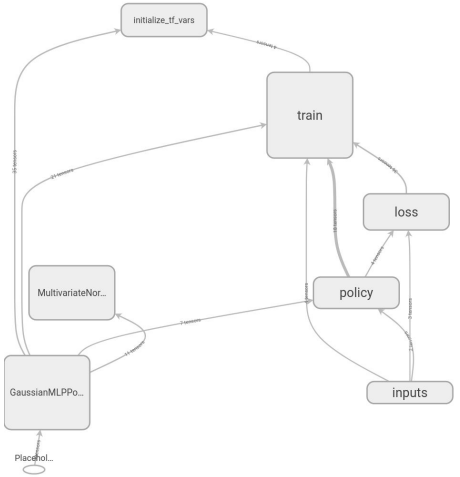
Introduction



Graph neural nets are showing great promise for particle tracking

See e.g. summary talk here

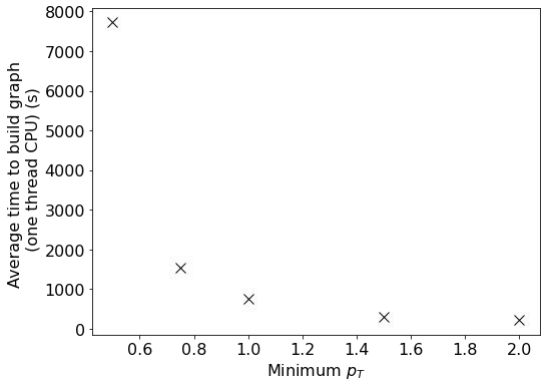
Graph building remains a challenge



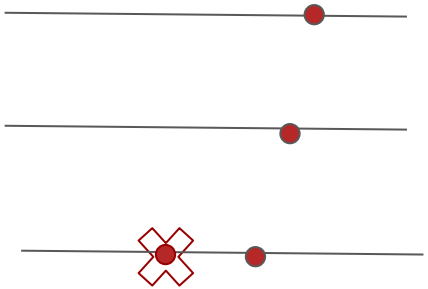
Seeded graph building and reinforcement learning are explored as options for accelerated graph building

Graph building is challenging

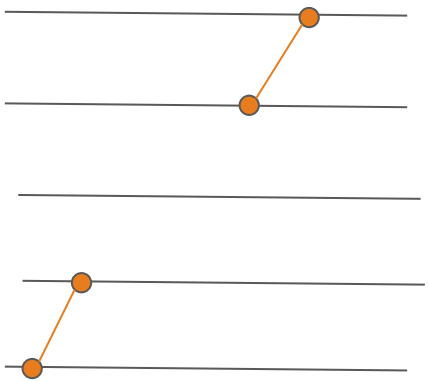
Graph building is slow



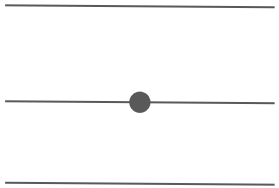
Allows only one hit per layer per track



Does not allow missing hits

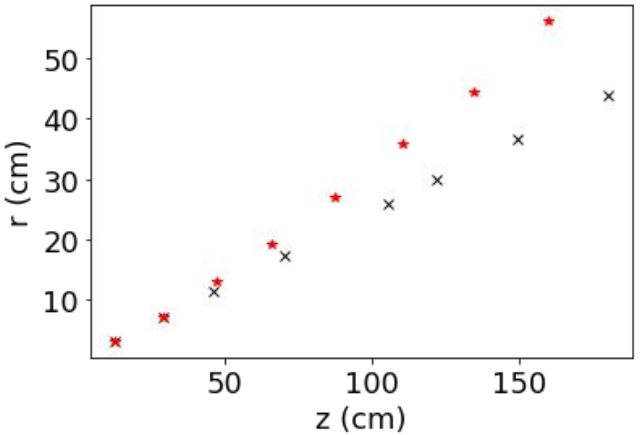


Filters out background hits



Motivation for reinforcement learning

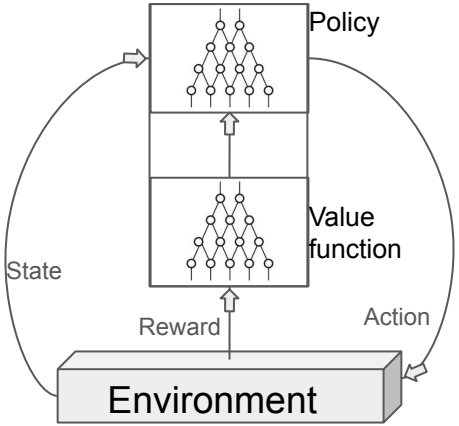
- Use experience to build smaller and simultaneously less limited graphs
- A straight line in r,z space should be easy to learn
- Can take advantage of physics information
- Easily parallelisable as each track is independent
- Freedom to correct incorrect behaviour

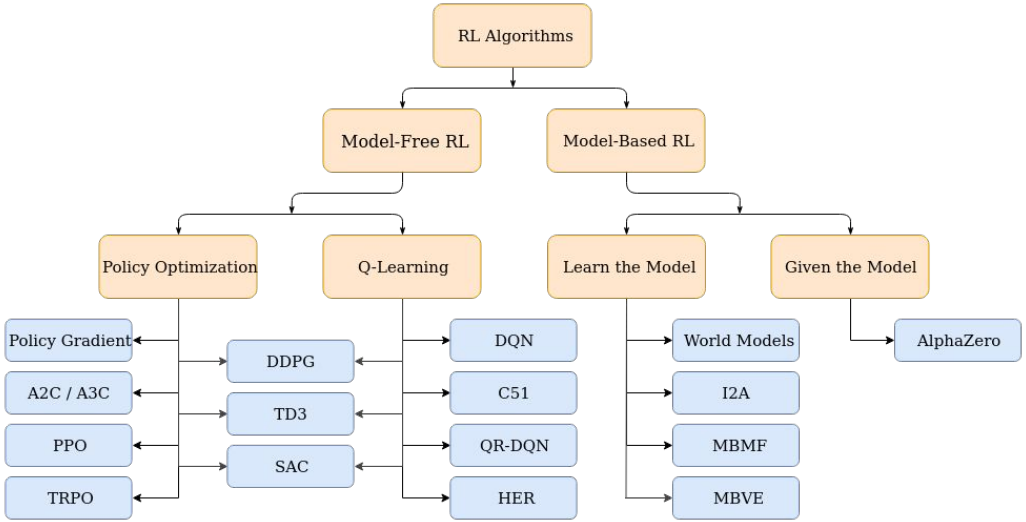


Reinforcement learning theory

An agent learns actions by rewards
“Requires” that:

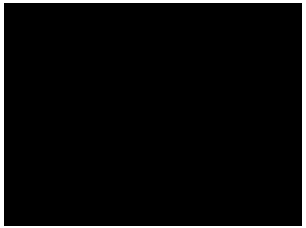
- Environment is intractable
- Agent receives data in response to action
- States are Markovian





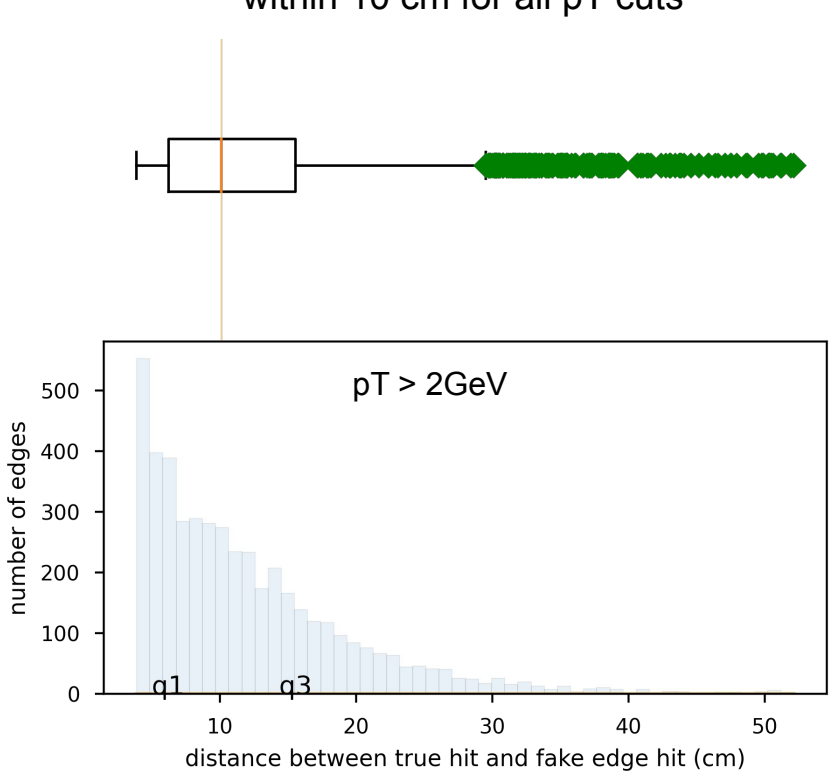
Successful applications

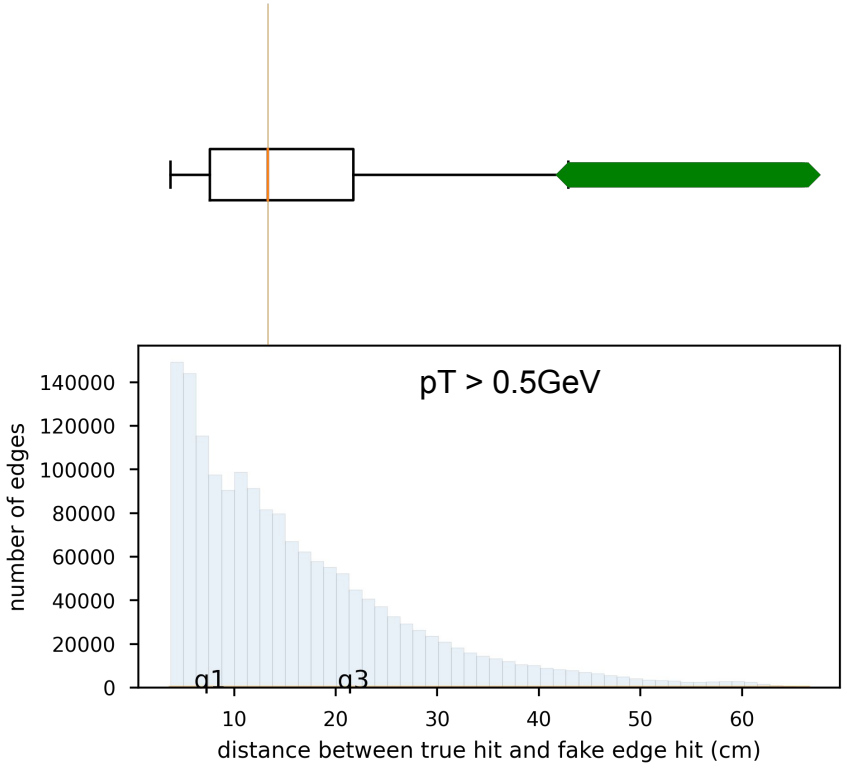
[Playing games]
[Walking robots]
[Controlling plasma]



Performance requirement estimate

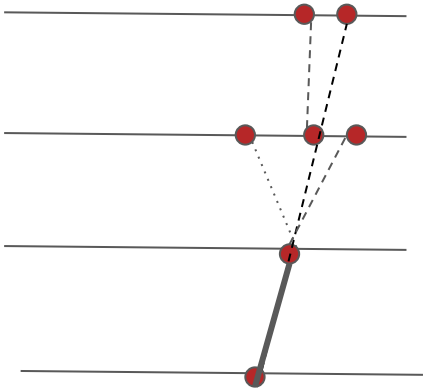
If RL model predicts *reliably* within 20 cm, ~25% of fake edges removed, ~50% if within 10 cm for all pT cuts





Reinforcement learning environment TrackML

Continue until n hits found



State described by position of hit

Reward is -distance between the correct hit and prediction

RL learns a correction to this assumption

Assume next position is position + Δr, Δz

Start with seed to find Δr, Δz

Vanilla policy gradient

Algorithm 1 Vanilla Policy Gradient Algorithm

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

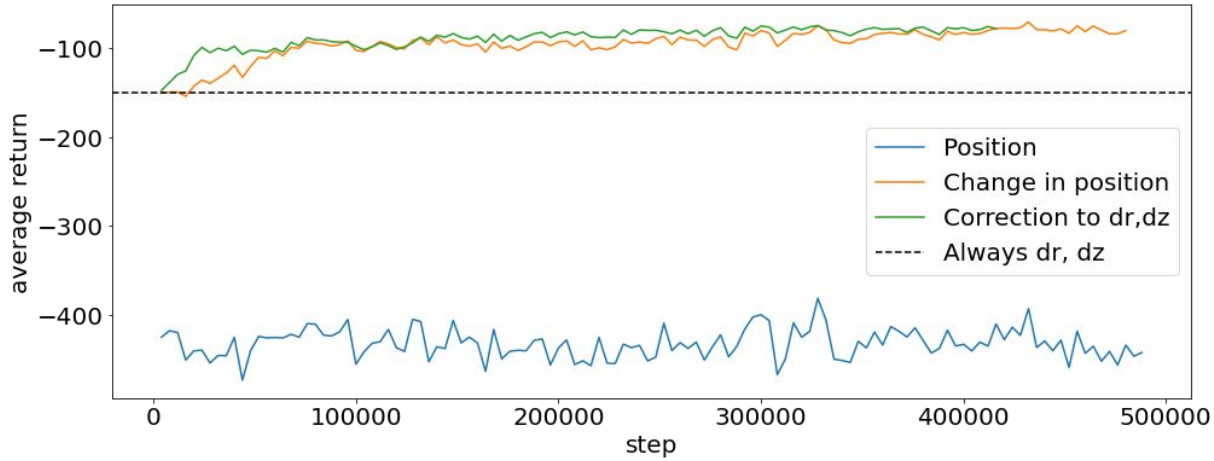
typically via some gradient descent algorithm.

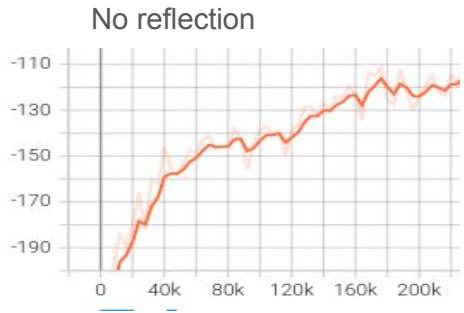
end for

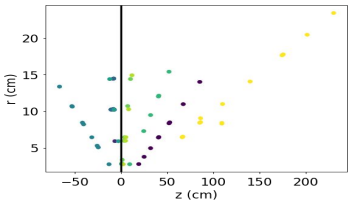
Inductive bias

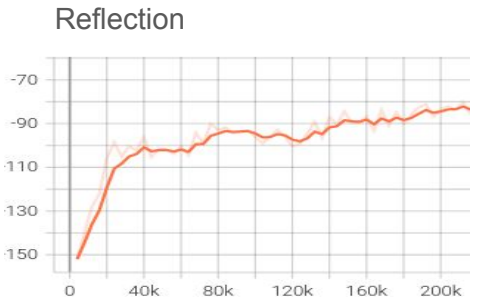
Use prior domain knowledge to improve accuracy:

- Prior assumption of next hit position
- Reflection in z space

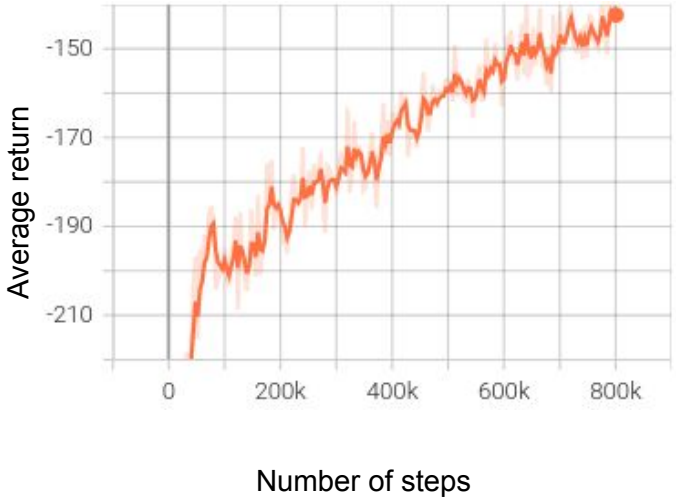






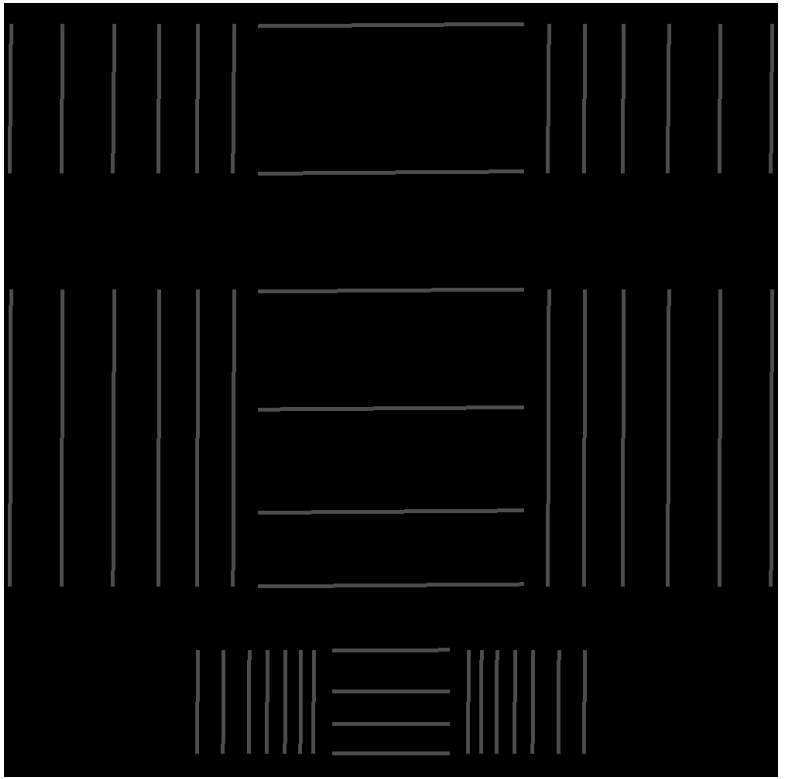


First pass result



True hits

RL predictions



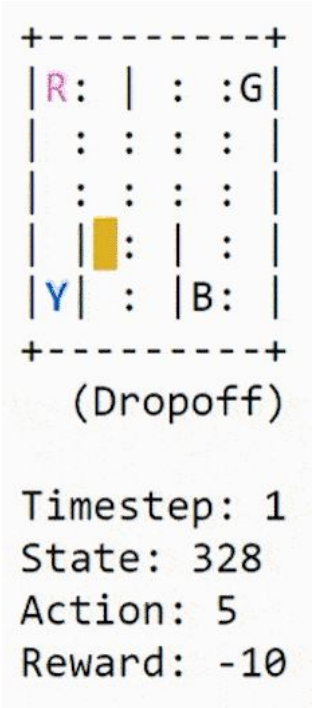
Q-learning for discrete states

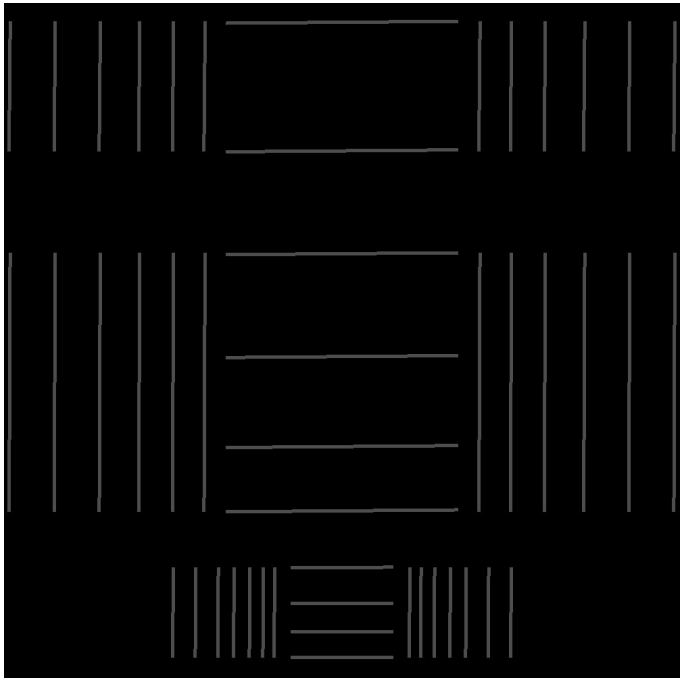
Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	...	...	...	...	...	...	...
	327	0	0	0	0	0	0
	...	...	...	...	...	...	...
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	...	...	...	...	...	...	...
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	...	...	...	...	...	...	...
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

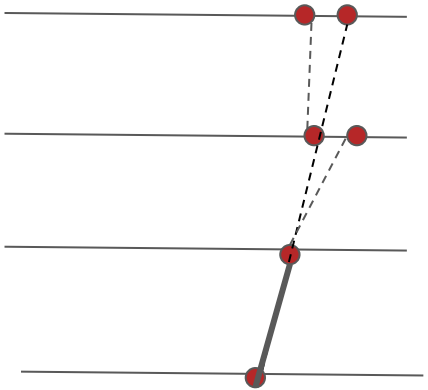




Figures from [1]

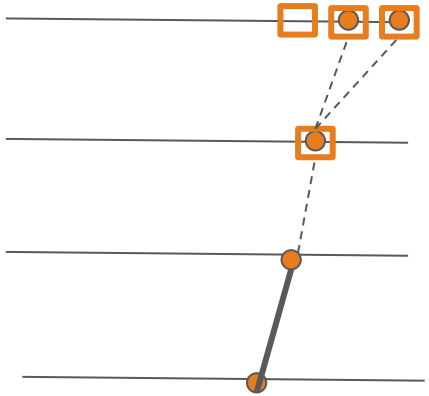
Seeded graph building

Prediction of intersection points



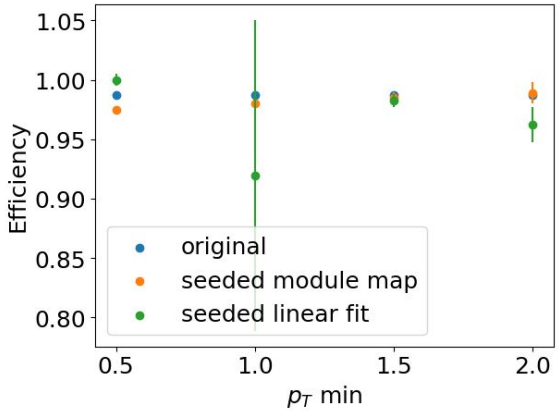
Create seed from two first hits sorted in r,z
Use seed to find line parameters
Predict intersection points with detector layers
Find hits close to intersection points

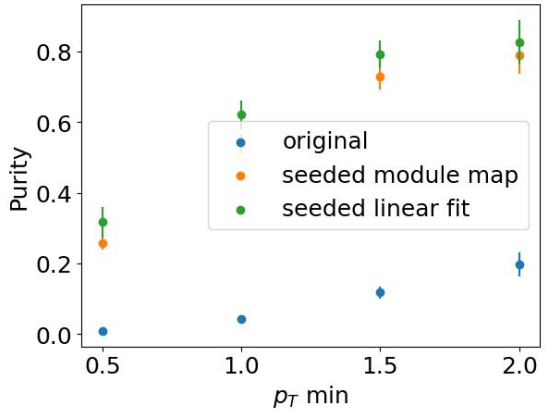
Straight line module map

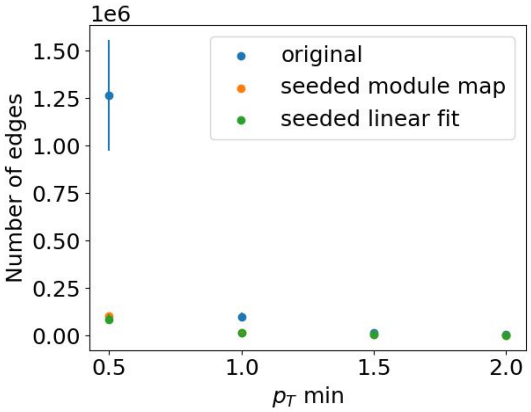


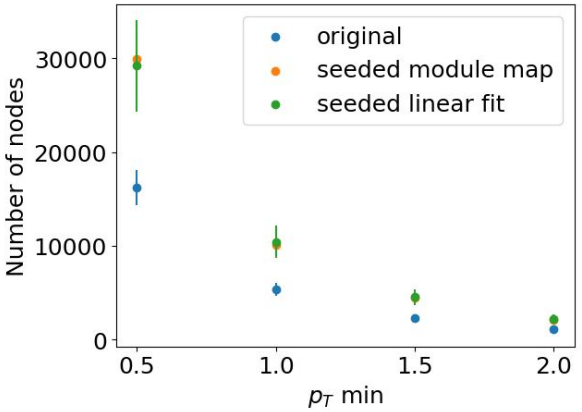
Create seed from two first hits sorted in r,z
Use seed to define line parameters
Find compatible modules given these parameters

Seeded graph building





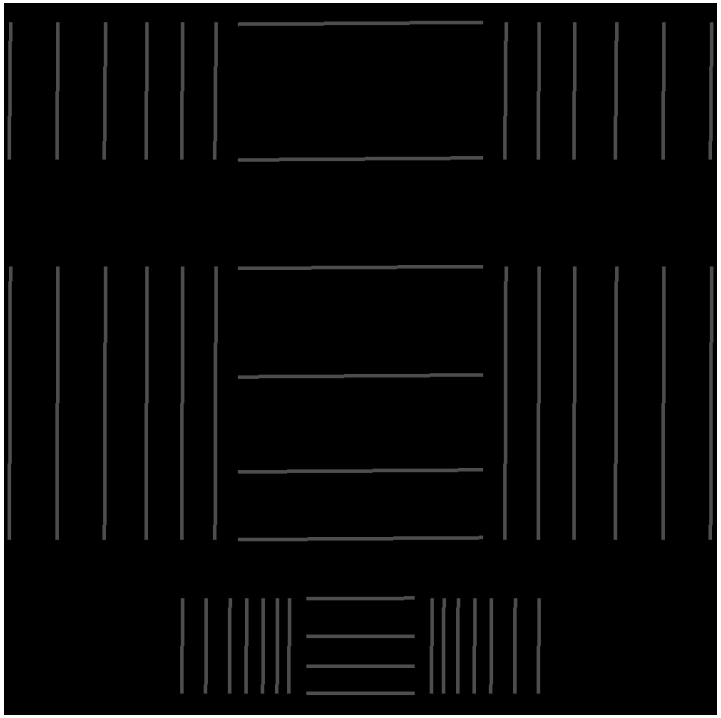




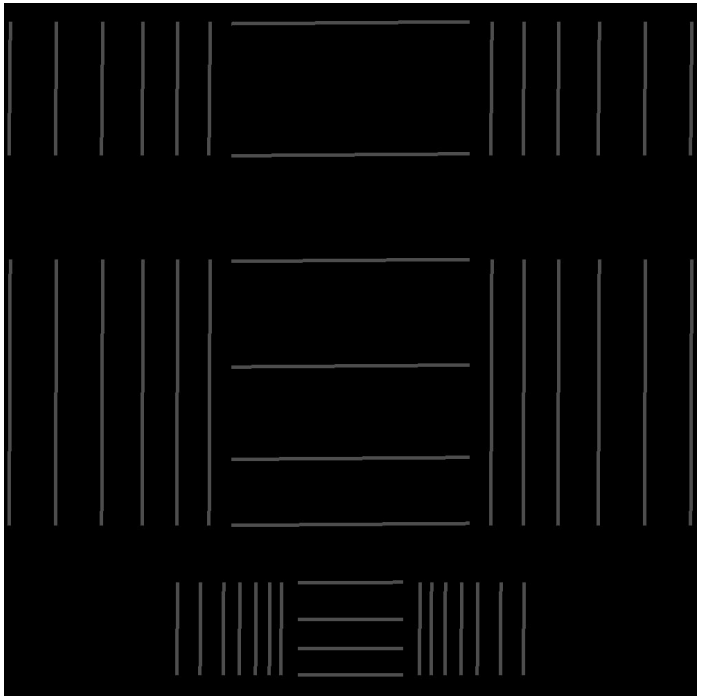
- Original is graph building as described in *Charged particle tracking via edge-classifying interaction networks*
- Seeds created from first two hits in track
- Using seeds leads to higher graph purity
- These methods are slow because they propagate one track at a time

Using seeded graph building methods with RL

No RL

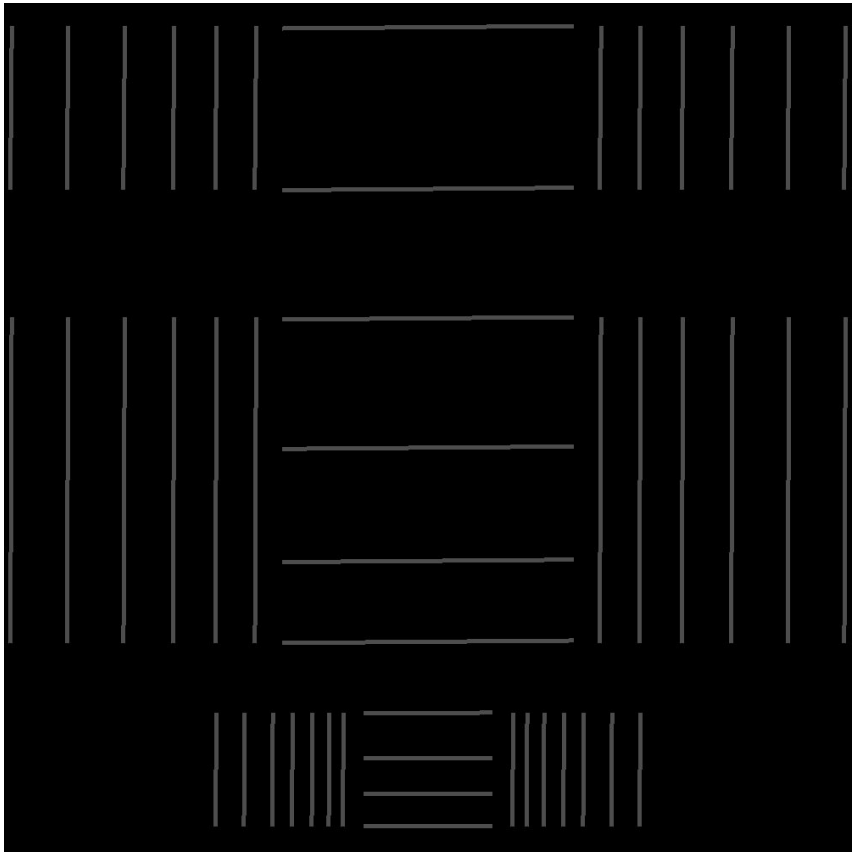


With RL



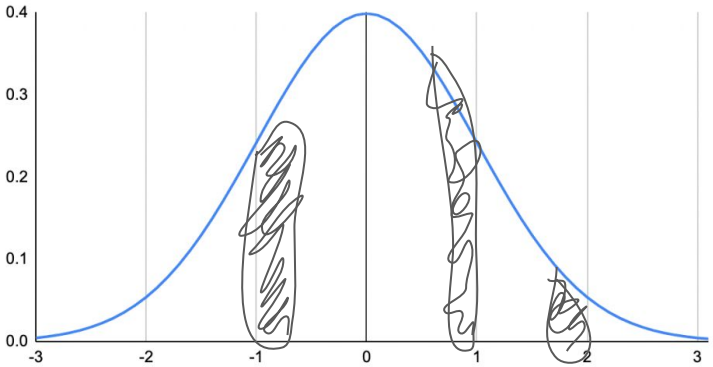
Unseeded reinforcement learning

Unseeded RL shows that action space is very unrestricted



Potential improvements

- Action masking - only sampling actions from allowed space
- Explore other RL methods and environments
- Include physics restrictions
- Combined learning of GNN and RF
  (see e.g. this talk using it for proton traces)



Conclusions

- Reinforcement learning can learn an approximation of the track hits
- This is so far not more accurate than using seeded intersection/module information
- Reinforcement learning should be more accurate than module mapping, so work will continue to explore this
- Potential candidate for FPGA acceleration