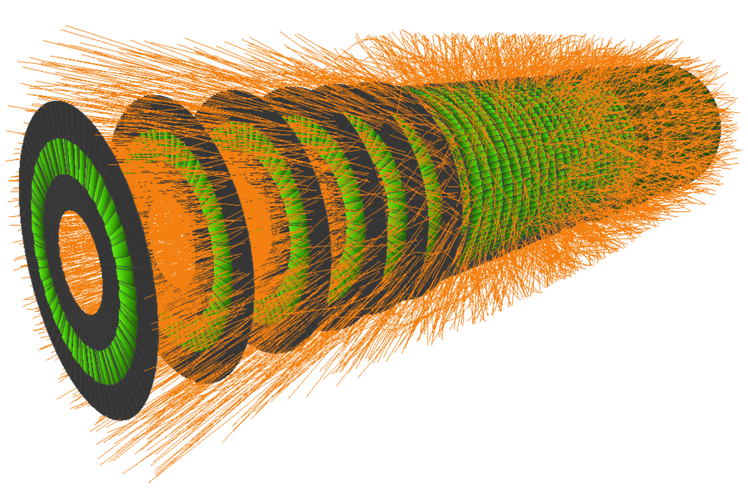
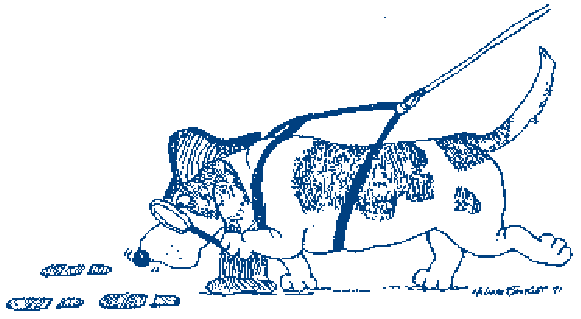




The FASTrack solution

Filter and Automaton for Silicon Tracking

Dmitry Emeliyanov

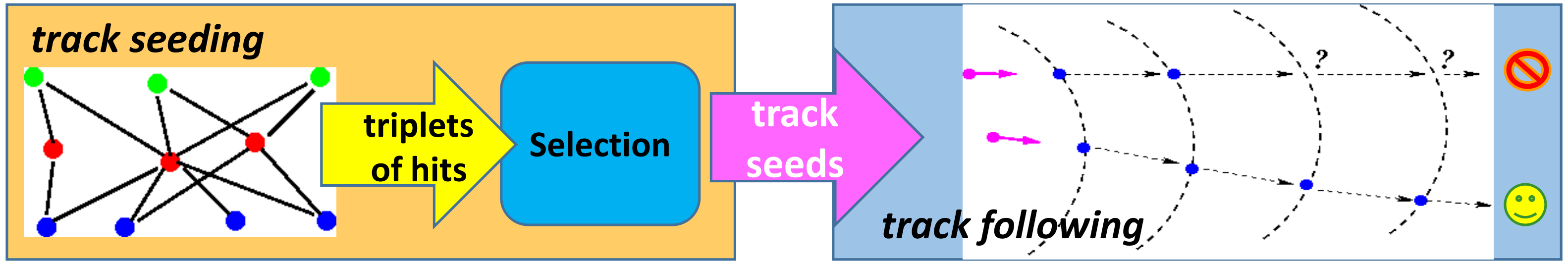
https://github.com/demelian/fastrack

Outline

- The state-of-the art approach and why it’s (relatively) slow
- The track finding methods and the FASTrack’s place among them
- The FASTrack solution: key ideas and techniques
  - track finding on graphs and its efficient implementation
  - Kalman filter and the magnetic field model
- Conclusion and outlook

The state-of-the-art

N seeds (PU80)		N tracks
Total	Accepted	
19750	4280	1220

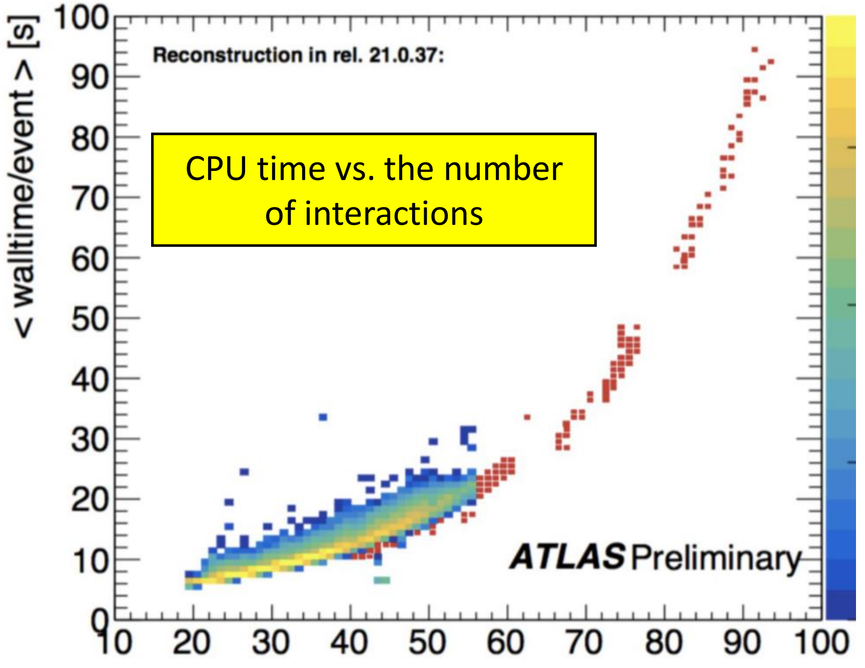


The CPU timing model :

$$T_{CPU} \sim T_{seed} + N_{seed} * T_{KF}$$

where $N_{seed} \sim N_{hits}^{\alpha}$ and $\alpha \approx 2.5$

- The Kalman filter-based track following is the main track finding algorithm used in ATLAS and CMS experiments
  - high track finding efficiency provided by the multi-pass seeding
  - Kalman filter accounts for various detector material effects providing precise track parameter estimates
- The problem with this approach is that its compute time scales non-linearly with hit multiplicity
  - determined the number of simultaneous pp collisions (PileUp)
  - For the future LHC upgrade with pileup levels up to 200 we will need x10 speed-up of the tracking in order to stay within the flat budget of computing resources



Classification of the track finding methods

- There are several groups of the track finding methods – the fundamental difference lies in the way how they define a track in the detector :

A track is

1

a sequence of hits in the detector

Sequential, “step-by-step”, data association e.g. the track following using Kalman filter

2

a point in the track parameter space or in a function space of possible track shapes

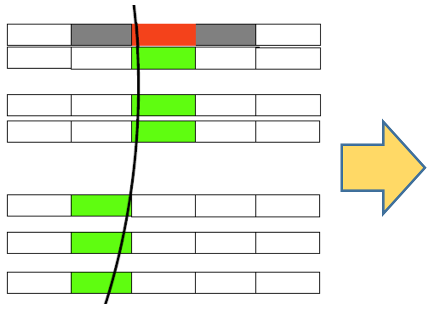
Pattern recognition using integral transforms (Radon, Hough), pattern (template) matching

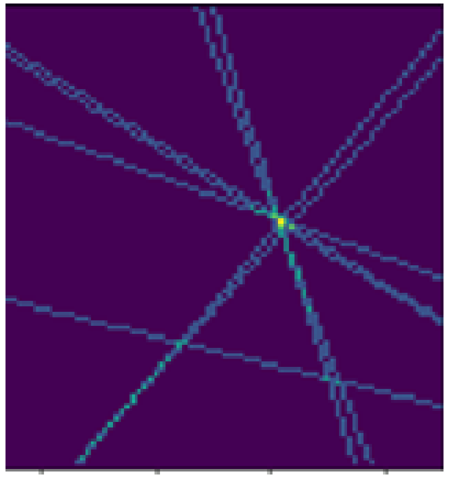
3

a sequence of track fragments = **points** in track parameter space

Track finding using cellular automata with graph-based track models

FASTrack





track parameter space

The FASTrack solution

Accuracy Phase score: 0.87

Throughput Phase:

Accuracy: 0.944
Time/event: 1.1 sec
Memory: 0.6 Gb

post-competition version:

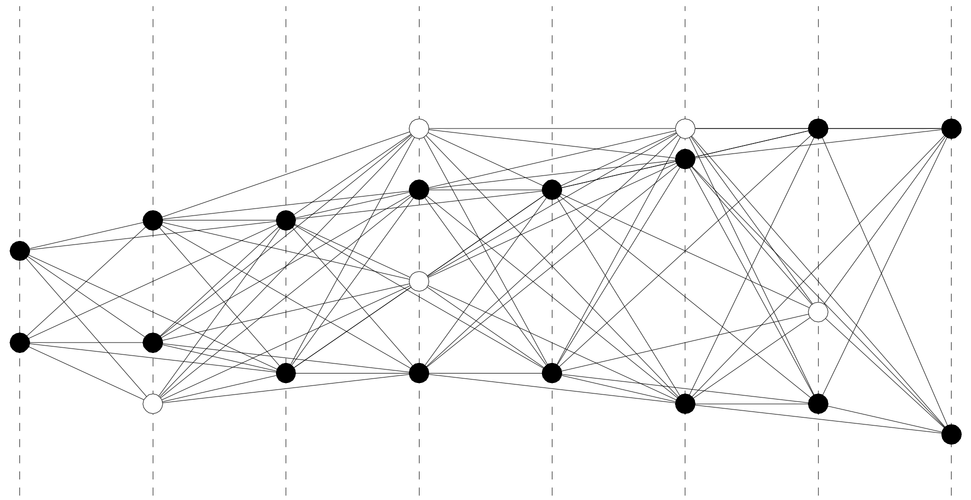
Accuracy: 0.948
Time/event: 0.8 sec

- combination of the graph-based track finding (GTF) and the standard Kalman filter-based track following (TF) :
  - the GTF algorithm discovers the bulk of the tracks
  - the TF puts "finishing touches": extends tracks to the layers not covered by the graph and fills the "holes" on tracks thus compensating the loss of hits if only a short fragment of a track is discovered by the GTF
- track finding is done in 3 passes with hit masking after each pass:
  - high-pT central tracks, low-pT central tracks, the rest of the tracks
  - formation of the hit-pair graph: hit pairs are vertices, connected hit pairs (i.e. triplets) are edges
    - the pairs of layers for hit pair formation are learned from data: track movement from layer to layer is treated as Markov process and state transition probabilities are measured using training data
  - using clusters shape to predict track inclination angles and avoid wrong hit pairs:
    - it was added for the Codalab Phase, greatly improved the accuracy and made the algorithm much faster
  - traversal of the hit-pair graph with an embedded Kalman filter for fast discovery of track candidates

Track finding on graphs

- General idea:
  - find short track segments, e.g. pairs or triplets of hits
  - create a graph of track segments by setting *admissible* connection between segments
  - a track is defined as an optimal (in some sense) path on the graph

- Example:
  - the segments connect hits
    - in the adjacent layers
    - across one layer
  - admissible connections:
    - segments have common hit
    - small breaking angle $\Delta\varphi(s_i, s_{i+1})$



Recursive track search on a graph

- Let $S_k = \{s_1, \ldots, s_k\}$ be a sequence of *connected* track segments
- Consider the following “best-path” criterion
  - the number of connected segments with imposed cut on the breaking angles

$$J(S_k) = \sum_{i=1}^{k-1} I\left(\left|\Delta\varphi\left(s_i, s_{i+1}\right)\right| \le \Phi\right), \quad I(x) = \begin{cases} 1, \text{if } x = \text{true} \\ 0, \text{if } x = \text{false} \end{cases}, \quad \Phi \text{ is a cut}$$

- $J^* = \max\limits_{S} J(S)$ satisfies Bellman recursion property:

$$J^*\left(S_{k+1}\right) = J^*\left(S_k\right) + \max_{s_{\mathrm{k}+1}} I\left(\left|\Delta\varphi\left(s_k, s_{k+1}\right)\right| \le \Phi\right)$$

  – so that optimal segment sequences (i.e. track candidates) can be constructed recursively by the dynamic programming algorithm

The cellular automaton approach

Track segments are called ***cells***, cell can have ***neighbours*** – connected segments, cell connections are oriented (i.e. from the inside of the tracker)
Each cell has a ***state*** – positive integer number

Historically, it was the first algorithm for track finding on graph models

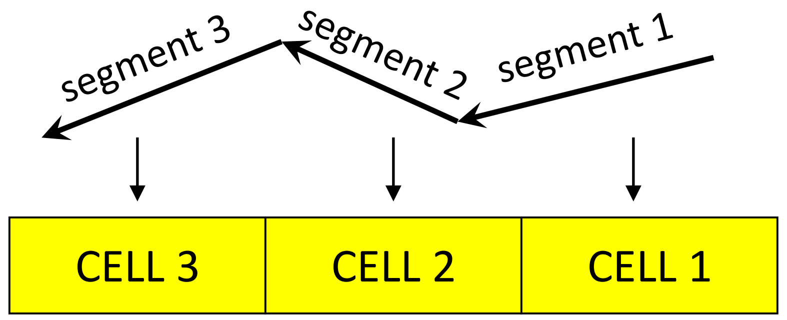
The algorithm works on the cell set and has two distinct phases :

- <u>Forward recursion</u>:

if cell $\boldsymbol{i}$ with the state $\boldsymbol{s_i}$ has a neighbour $\boldsymbol{j}$ with the equal state $\boldsymbol{s_j} = \boldsymbol{s_i}$ then, at the next iteration, the state is incremented:

$$\boldsymbol{s_i} = \boldsymbol{s_j} + 1$$

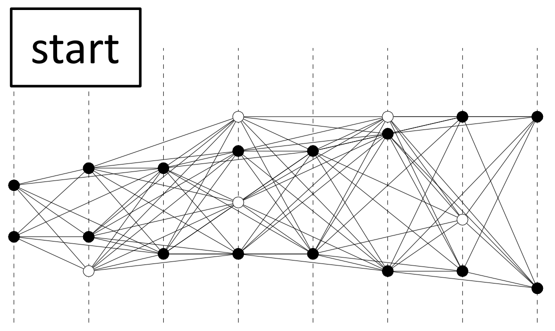
The iterations stop when there is no neighbouring cells with equal states. The final cell state is equal to the length of the segment sequence which can be traced to the left starting from this segment

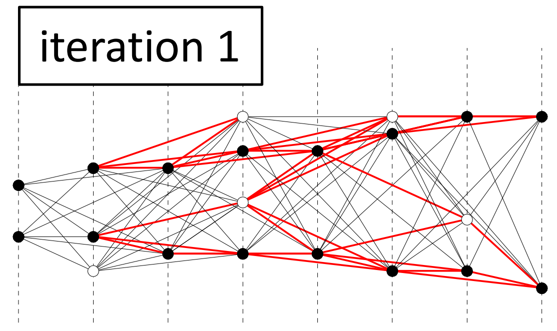


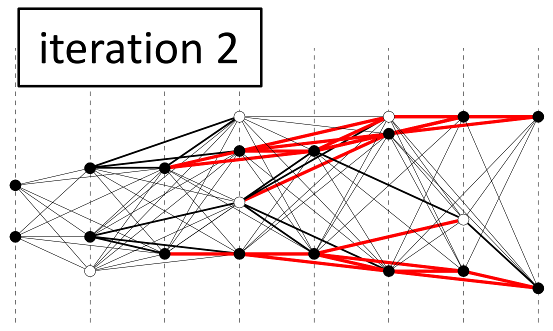
Iter 0	s=1	s=1	s=1
Iter 1	s=1	s=2	s=2
Iter 2	s=1	s=2	s=3

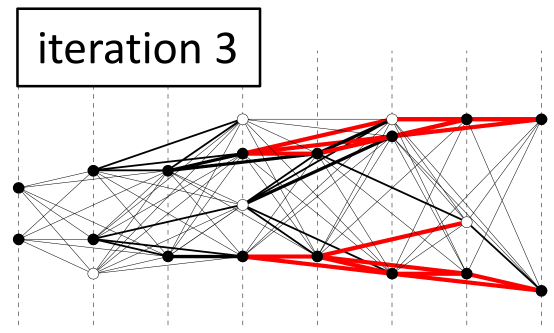
- <u>Backward pass</u>: Find the cell with the highest state $\boldsymbol{S}$, then find its neighbour with $\boldsymbol{S\text{-}1}$ and so on until state=1. If there are a few neighbours with the same state, take the segment with the smallest breaking angle. The backward pass is repeated the next best cell/state until all the track candidates starting from segments above the threshold are collected

CA in action

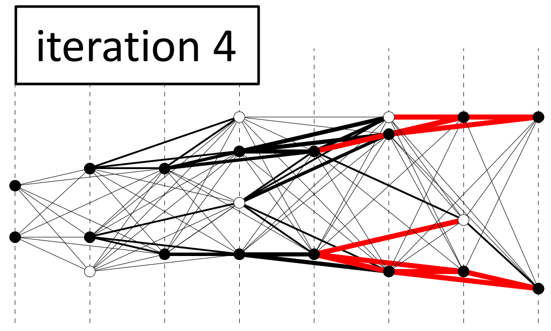


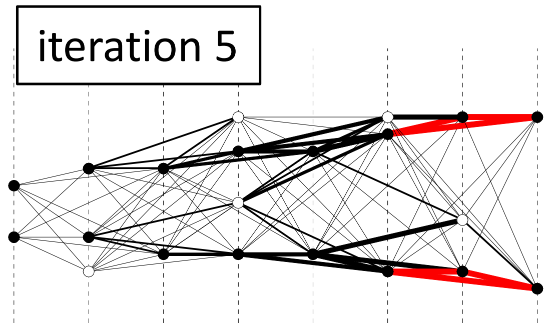


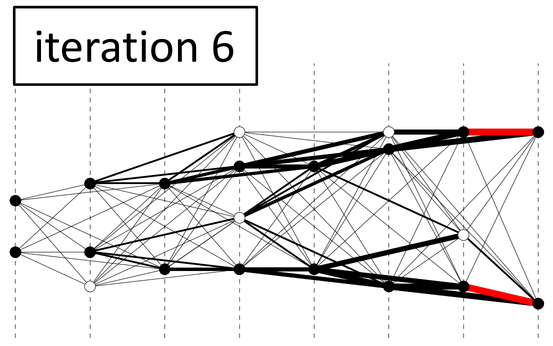


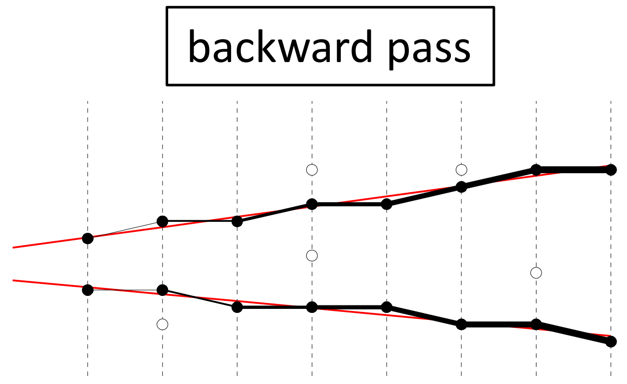


Line width indicates a cell state, color Red: state updated at the iteration, **Black**: no state update









Efficient implementation

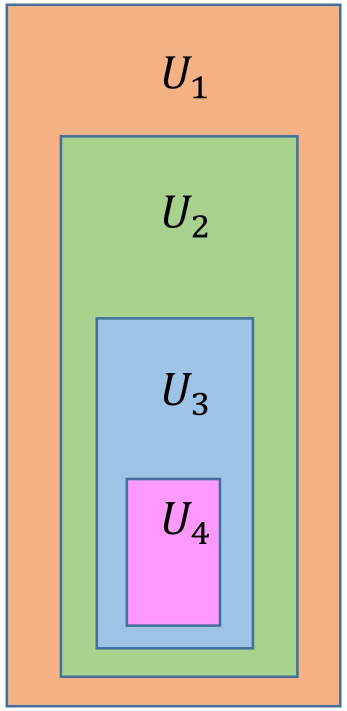
- Due to the large number of cell state look-ups the CA-based implementation of the forward recursion is computationally inefficient
- More efficient is to use the Dynamic Programming (DP) recursion :
  - we maintain the set of cells U_i updated at the $i-th$ iteration
  - it's easy to show that the following inclusion takes place:

$$U_{i+1} \subset U_i \subset U_{i-1} \subset U_{i-2} \dots ,$$

so that no cells outside the set U_i will be updated at the next iteration

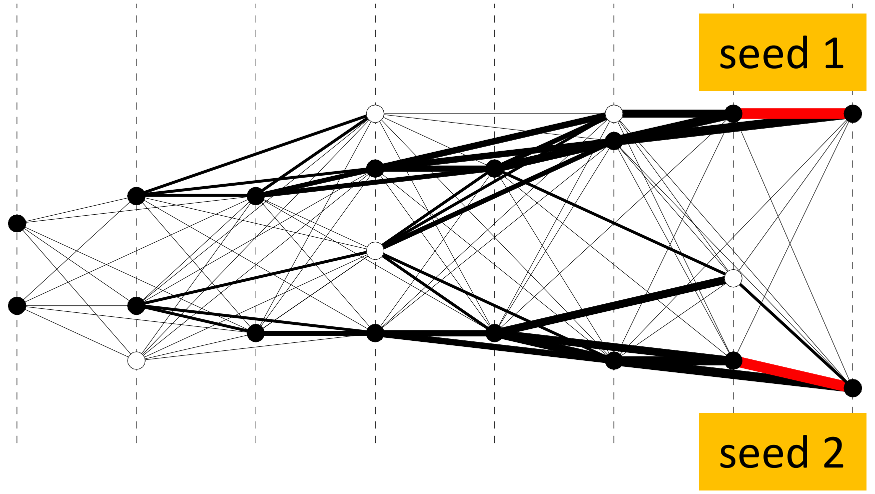
  - therefore only the cells updated at the current iteration must be checked for possible updates at the next iteration
- After replacing the CA with the above computational scheme the FASTrack CPU time was reduced by 30% from 1.1 s to 0.8 s

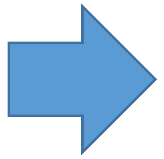
The subsets of updated cells

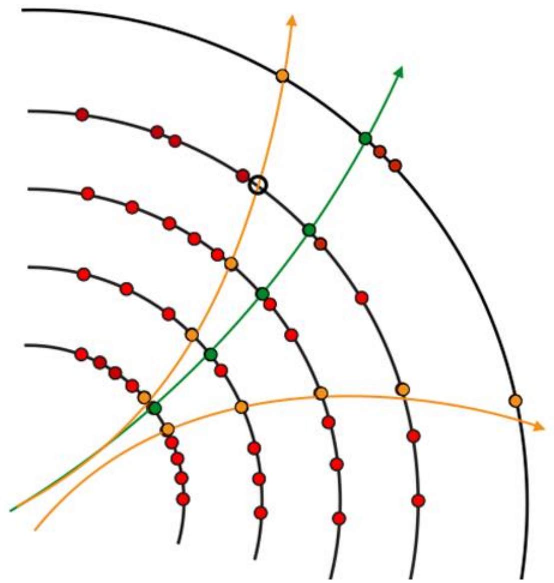


Track following on a graph

- The idea is to embed Kalman filter in the backward pass of the graph-based tracker :
  - hence the name “Filter and Automaton for Silicon Tracking” : FASTrack
  - it’s a crude track following which operates on connected track segments rather than on hits
  - works with “seeds” which are guaranteed to lead to good track candidates thus saving CPU time
- The filter tracks (φ,θ) angles rather than positions using simple dynamics model :
  - random walk for the r-z plane (non-bending), AR(1) model for the r-phi plane (bending) – fitting a trend in φ evolution along a track rotating in the magnetic field



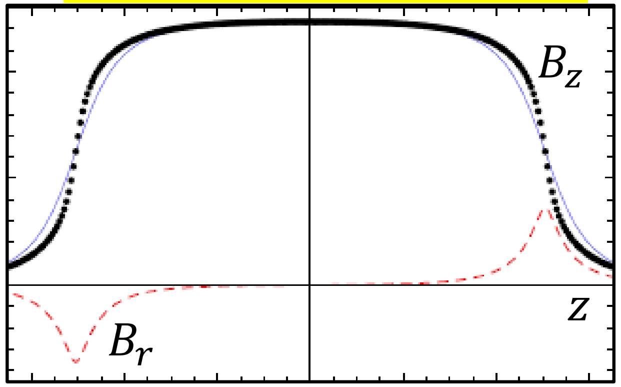


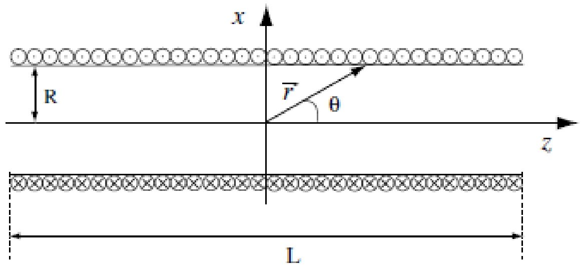


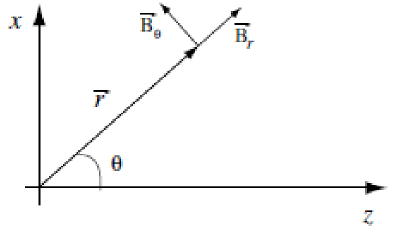
The Kalman filter and magnetic field model

- Using the model derived in https://arxiv.org/abs/1003.3720
  - the third order approximation (w.r.t. $\cos\theta$) for the field inside a finite solenoid
  - free parameters: field at (0,0,0), solenoid length L, solenoid aspect ratio (R/L) – learned from data by maximizing accuracy score via parameter scan
- Track parameter and covariance extrapolation in the Kalman Filter
  - the KF is used for track extension to the long strip volumes (16,17,18) and for precise track fitting which calculates χ^2 and track likelihood
  - the extrapolation between two detector planes requires solving the boundary value problem (BVP) as the extrapolation step depends on track parameters and field and cannot be calculated analytically
  - the BVP is solved by the shooting method with 3rd order Runge-Kutta integrator

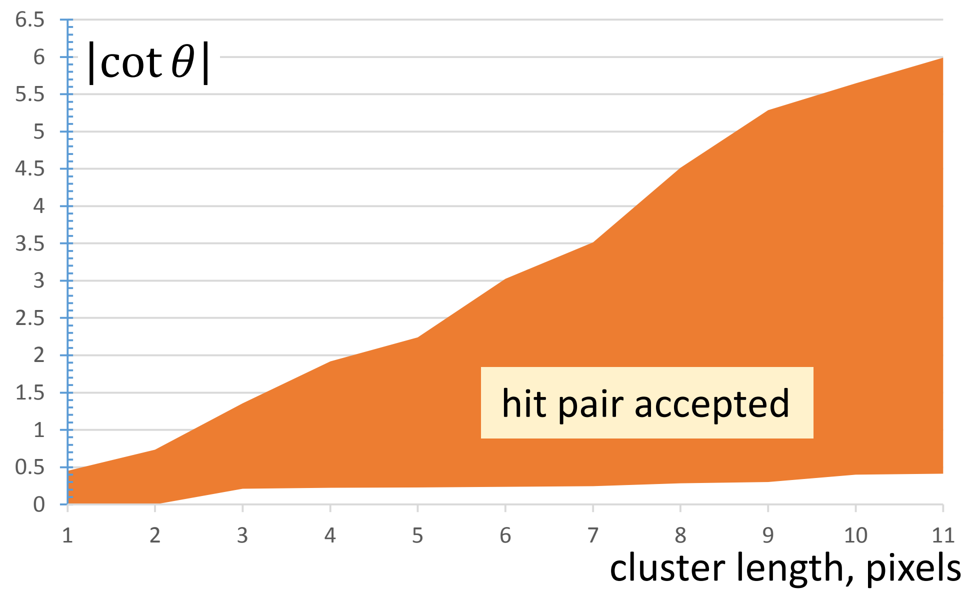
Field inside a finite solenoid

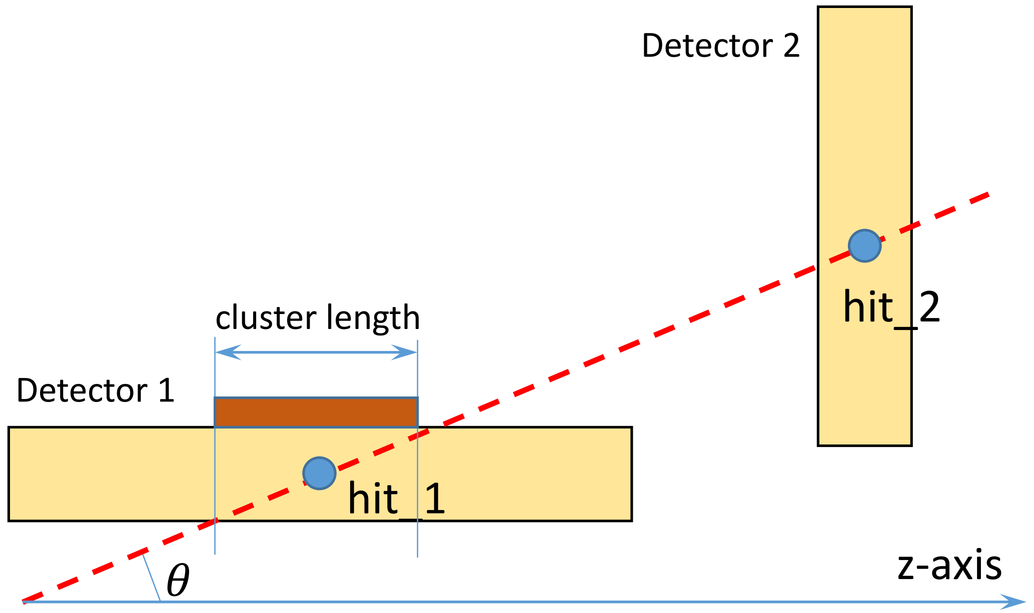






FASTrack : hit pair prediction





- Due to huge number of hit pairs (up to 1.5 million) the pair creation and the graph building are the most time consuming parts of the FASTrack algorithm
  - similar to the “Top Quarks” and “outrunner” solutions the FASTrack uses cluster shapes to predict if two hits belong to the same track in order to reduce the number of wrong pairs
  - min/max $|\cot\theta|$ as functions of cluster width were extracted from training data
  - for faster prediction the decision function was approximated by a LookUp Table (LUT)

Conclusion and outlook

- The FASTrack demonstrated the feasibility of combining
  - the fast track discovery using graph models
  - the Kalman filter-based track following
- Further algorithm improvements could include
  - z-vertexing: detecting interaction vertices – origins of the tracks <u>before</u> the actual track finding
  - integration of the z-vertexing with the graph formation/track finding process
  - staged graph formation/forward recursion driven by a layer-pair DAG trained on data
- Looking further ahead:
  - As we have seen, some of the best solutions employ the multi-pass track finding :
    - they disentangle the combinatorial puzzle of data association starting with "easy" tracks and gradually reducing the amount of data while progressing towards more complex cases
    - it more like progressive "explaining data away" rather than just labelling the hits
  - Ultimate goal for further research (as far as I can see it) :
    - development of some reasonably fast (e.g. hardware-accelerated) AI-like solutions which could reproduce the above type of approach/reasoning

The End

Thanks to all organizers and participants for the interesting challenge!