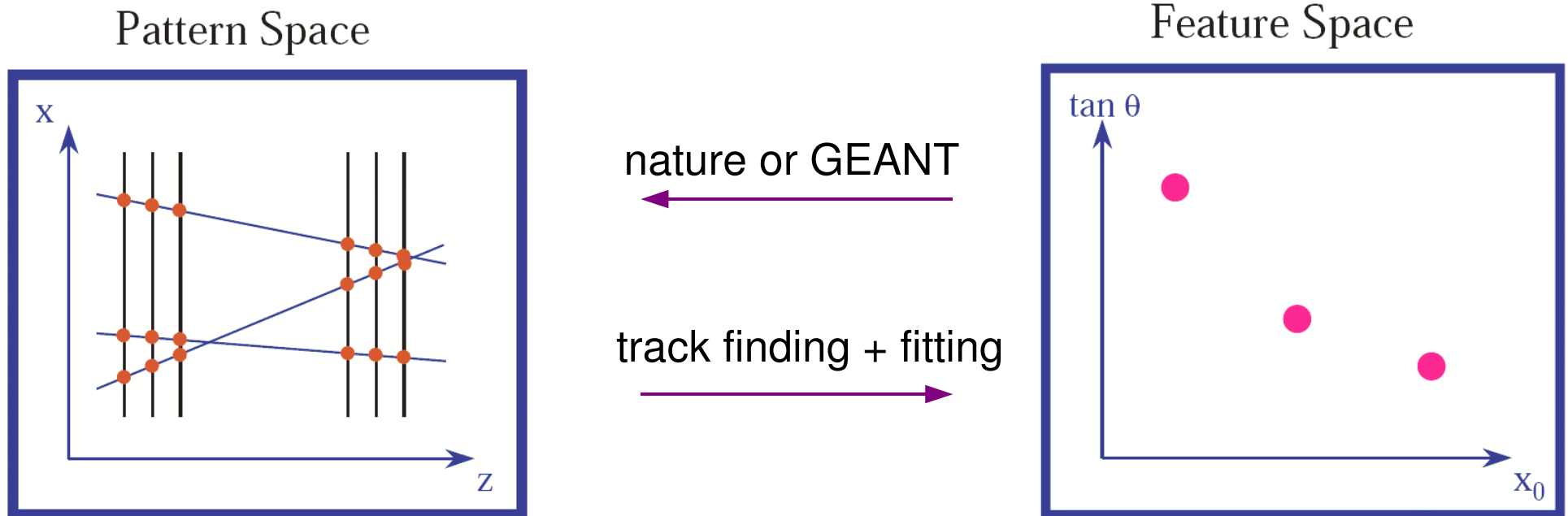


Part 5

pattern recognition

pattern recognition

- track pattern recognition: associate hits that belong to one particle



- will discuss concepts and some examples
- if you are interested in this, start with

R. Mankel, "*Pattern Recognition and Event Reconstruction in Particle Physics Experiments*",
Rept.Prog.Phys.67:553,2004, arXiv: physics/0402039.

aim of track finding algorithm

- two distinct cases
 1. reconstruct complete event, as many 'physics' tracks as possible
 - common for 'offline' reconstruction
 2. search only for subset of tracks, for example
 - in region of interest seeded by calorimeter cluster
 - above certain momentum threshold

typical in online event selection (trigger)
- how do we judge performance of algorithms?

efficiency

- **track finding efficiency**: what fraction of *true* particles has been found?
- two common definitions
 - by hit matching: particle found if certain fraction of hits correctly associated
 - by parameter matching: particle found if there is a reconstructed track sufficiently close in feature space
- usually need some criterion to decide if true track is 'reconstructable'

$$\text{total efficiency} = \text{geometric efficiency} \times \text{reconstruction efficiency}$$
- needless to say, track finding algorithms aim for high efficiency

ghosts and clones

- **ghost track:** reconstructed track that does not match to true particle, e.g.
 - tracks made up entirely of noise hits
 - tracks with hits from different particles
- **track clones:** particles reconstructed more than once, e.g.
 - due to a kink in the track
 - due to using two algorithms that find same tracks
- tracking algorithms need to balance efficiency against ghosts/clone rate
 - required purity of selection might depend on physics analysis
 - when comparing different algorithms, always look at both efficiency and ghost/clone rate

multiplicity and combinatorics

- **multiplicity**: number of particles or hits per event
 - central issue in pattern recognition: if there were only one particle in the event, we wouldn't have this discussion
- large multiplicity can lead to large **occupancy**, resulting in e.g.
 - overlapping tracks --> inefficiency
 - ghost tracks

to keep occupancy low, we need high detector **granularity**

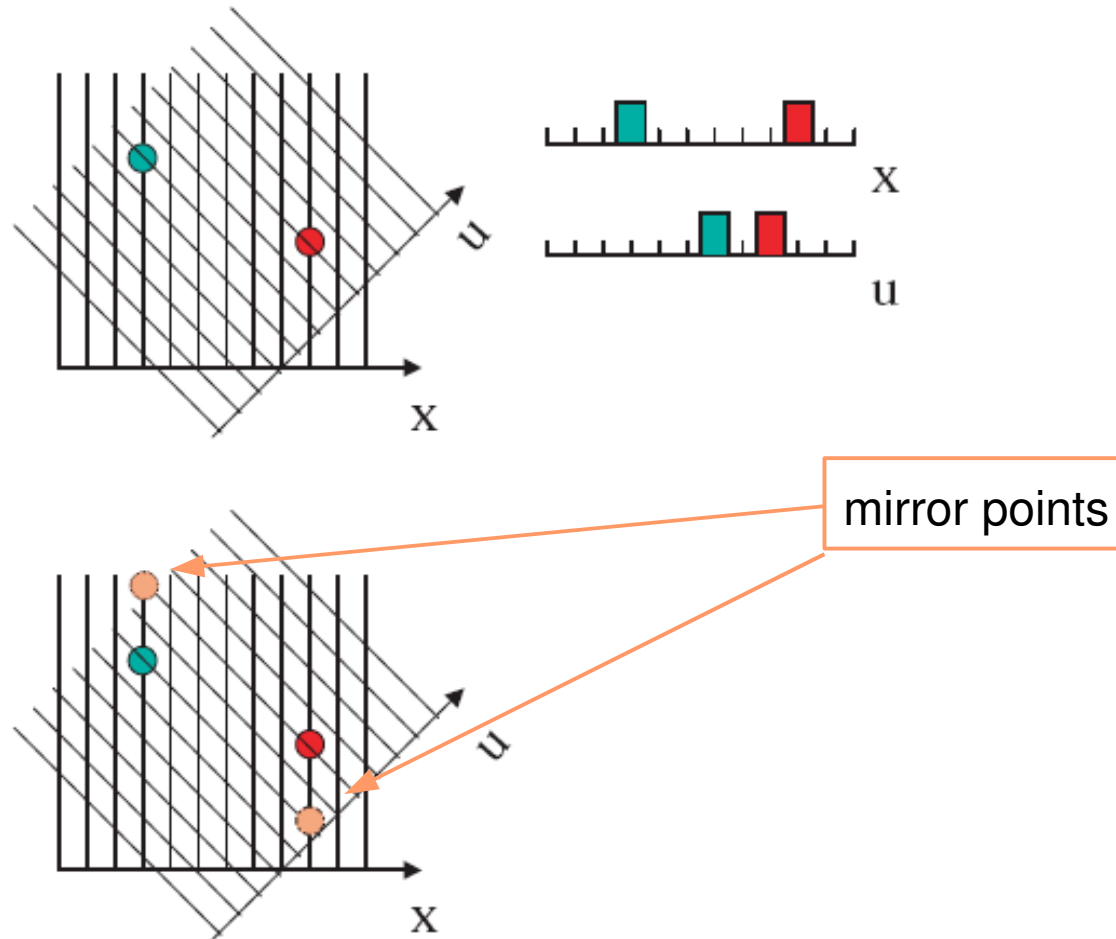
- large multiplicity also leads to large **combinatorics** in track finding
 - this is where algorithms become slow
 - usually, faster algorithms are simply those that try 'less combinations'
 - good algorithms are robust against large variations in multiplicity

2D versus 3D track finding

- single-coordinate detectors (like strip and wire chambers)
 - require stereo angles for 3D reconstruction
 - geometry often suitable for reconstruction in **2D projections**
- reconstruction in 2D projection reduces combinatorics
 - many track finding techniques only work in 2D
 - find tracks in one view first, then combine with hits in other views, or
 - find tracks in two projections, then combine
- 3D algorithms usually require 3D 'points' as input
 - need **space-point** reconstruction by combining stereo views
 - in single-coordinate detectors this leads to space-point ambiguity

space points ambiguity

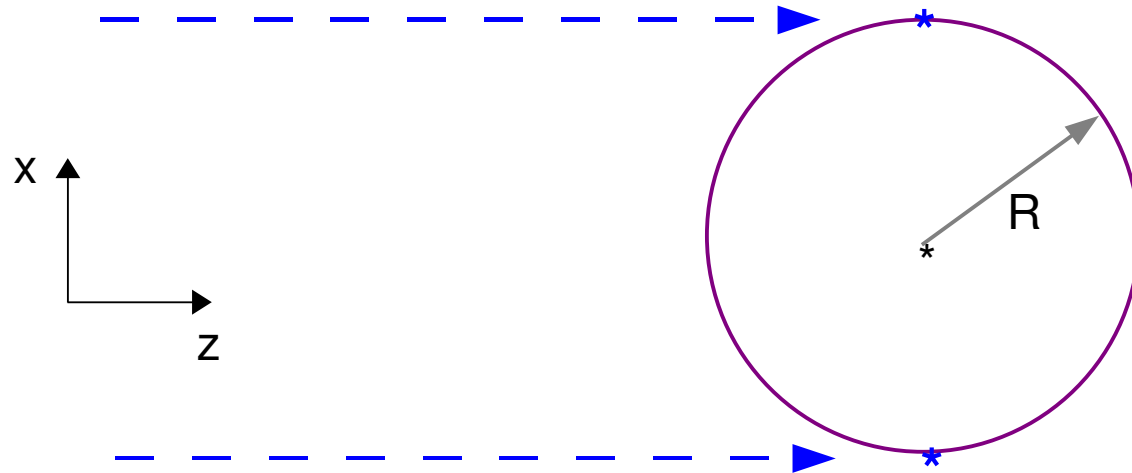
- consider 'x' and 'u' view at 45°



- problem worse if angle larger (since more strips overlap)
- need 3 stereo views to resolve ambiguities

left-right ambiguity

- drift-radius measurement yields 'circle' in plane perpendicular to wire
- leads to two possible hit positions in x-projection



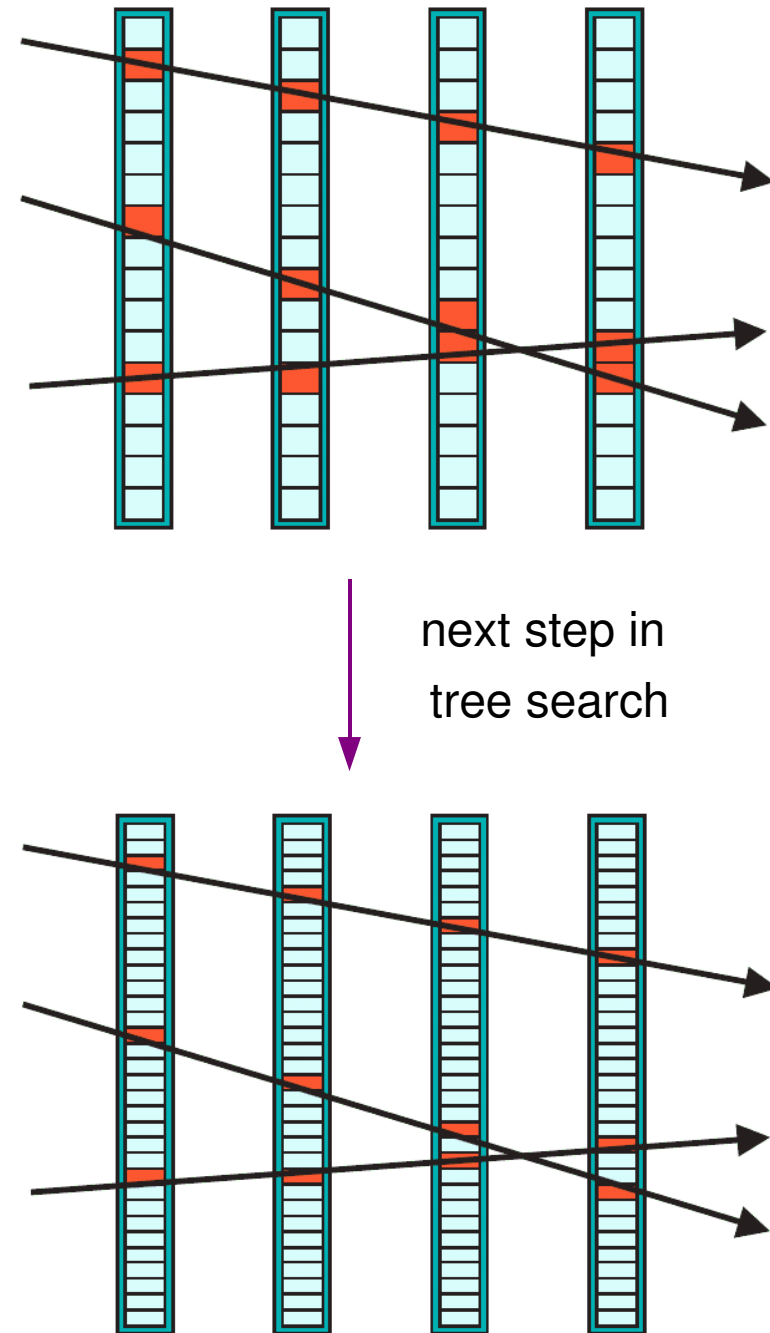
- this is called left-right ambiguity
- alternative way of thinking about this: two 'minima' in hit chi-square contribution (strongly non-linear)
- pattern recognition includes also 'solving' left-right ambiguities

track finding strategies: global versus local

- global methods
 - treat hits in all detector layers simultaneously
 - find all tracks simultaneously
 - result independent of starting point or order of hits
 - examples: template matching, hough transforms, neural nets
- local methods ('track following')
 - start with construction of track seeds
 - add hits by following each seed through detector layers
 - eventually improve seed after each hits (e.g. with Kalman filter technique)

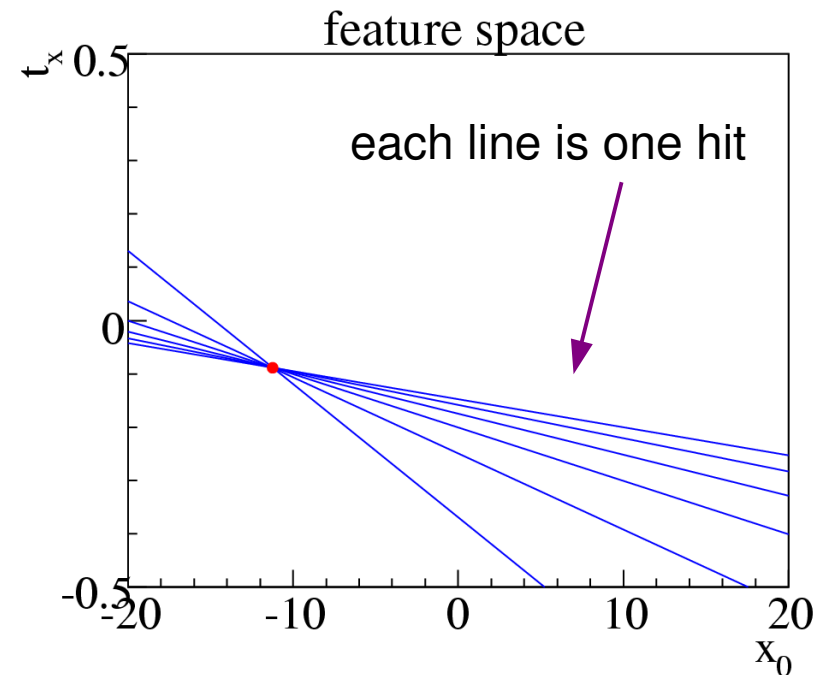
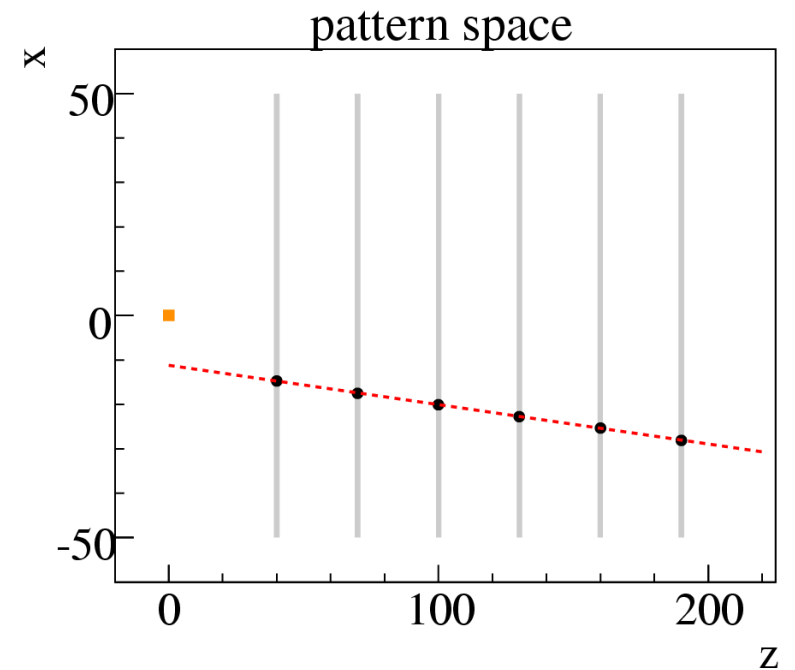
template matching

- make complete list of 'patterns', valid combinations of hits
- now simply run through list of patterns and check for each if it exists in data
- this works well if
 - number of patterns is small
 - hit efficiency close to one
 - simple geometry, e.g. 2D, symmetric, etc
- for high granularity, use 'tree search':
 - start with patterns in coarse resolution
 - for found patterns, process higher granularity 'daughter-patterns'



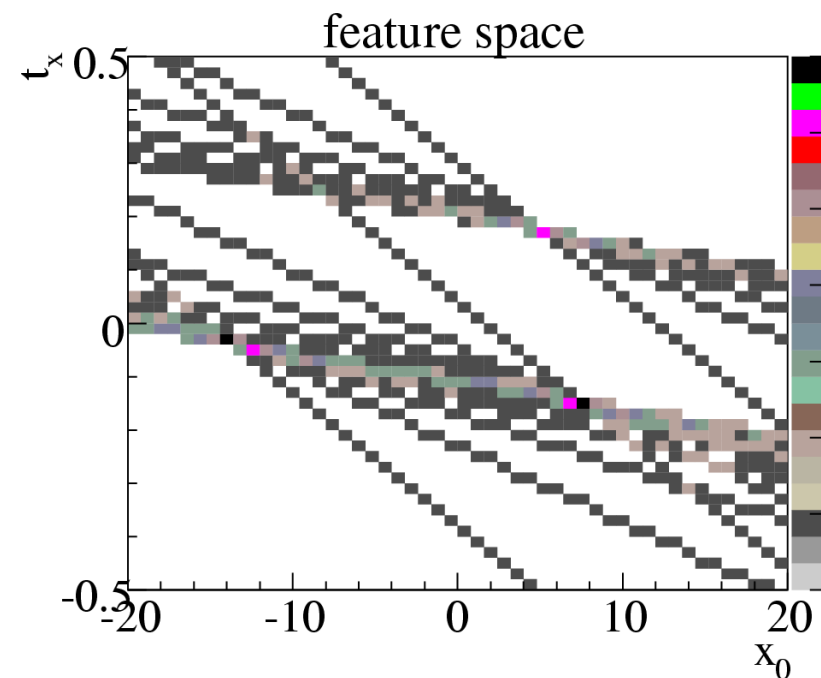
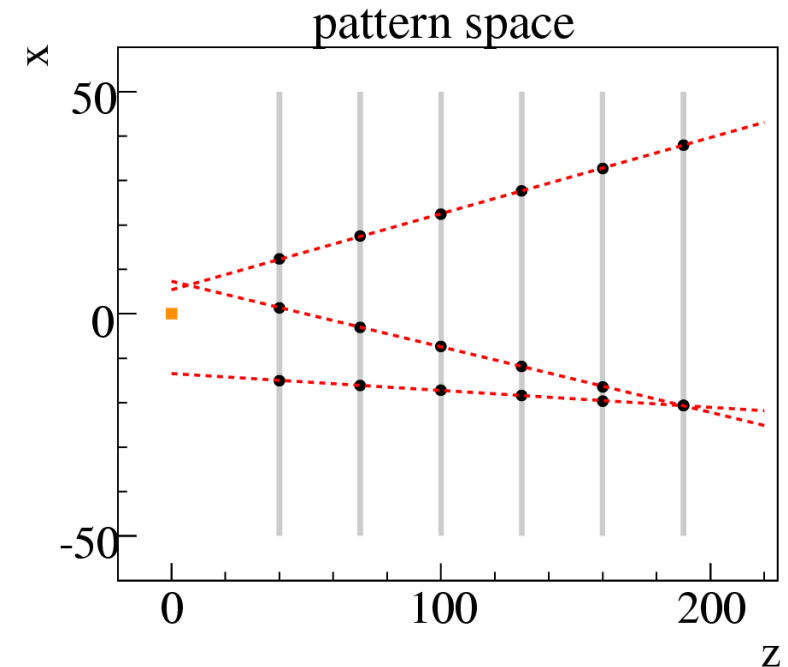
Hough transform

- hough transform in 2D:
point in pattern space -->
line in feature space
- example in our toy-detector
hit (x, z) --> line $t_x = (x - x_0) / z$
- lines cross at parameters of track
- plot on the right is for 'perfect resolution'



Hough transform (II)

- in real applications: finite resolution, more than one track
- concrete implementation
 - histogram the 'lines'
 - tracks are local maxima or bins with $\geq N$ entries
- works also in higher dimension feature space (e.g. add momentum), but finding maxima becomes more complicated (and time consuming)
- can also be used in cylindrical detectors: use transform that translates circles into points

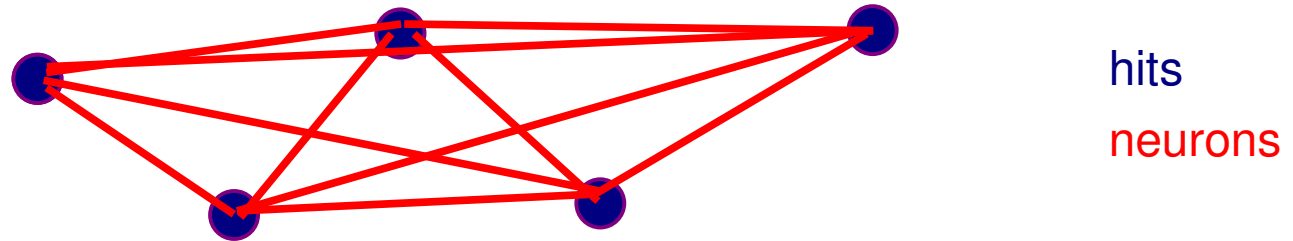


artificial neural network techniques

- ANN algorithms look for global patterns using local (neighbour) rules
 - build a network of neurons, each with activation state S
 - update neuron state based on state of connected neurons
 - iterate until things converge
- exploited models are very different, for example
 - Denby-Peterson: neurons are things that connect hits to hits
 - elastic arms: neurons are things that connect hits to track templates
- main feature: usually robust against noise and inefficiency
- we'll discuss two examples

Denby-Peterson neural net

- in 2D, connect hits by lines that represent binary neurons



- neuron has two different states:
 - $S_{ij} = 1$ if two hits belong to same track
 - $S_{ij} = 0$ if two hits belong to different tracks
- now define an 'energy' function that depends on things like
 - angle between connected neurons: in true tracks neurons parallel
 - how many neurons: number of neurons \sim number of hits
- track finding becomes 'minimizing energy function'

Denby-Peterson neural net

- energy function in the Denby-Peterson neural net

$$E = -\frac{1}{2} \sum \frac{-\cos^m \theta_{ijk}}{d_{ij} + d_{jk}} S_{ij} S_{jk} + \frac{\alpha}{2} \left(\sum_{l \neq j} S_{ij} S_{il} + \sum_{k \neq i} S_{ij} S_{kj} \right) + \frac{\delta}{2} \left(\sum S_{kl} - N \right)^2$$

'cost function':

- θ_{ijl} : angle between neurons ij and jl
- d_{ij} : length of neuron ij

penalty function
against bifurcations

penalty function to balance number of
active neurons against number of hits

- alpha, delta and m are adjustable parameters
 - weigh the different contributions to the energy
 - that's what you tune on your simulation
- minimize energy with respect to all possible combinations of neuron states

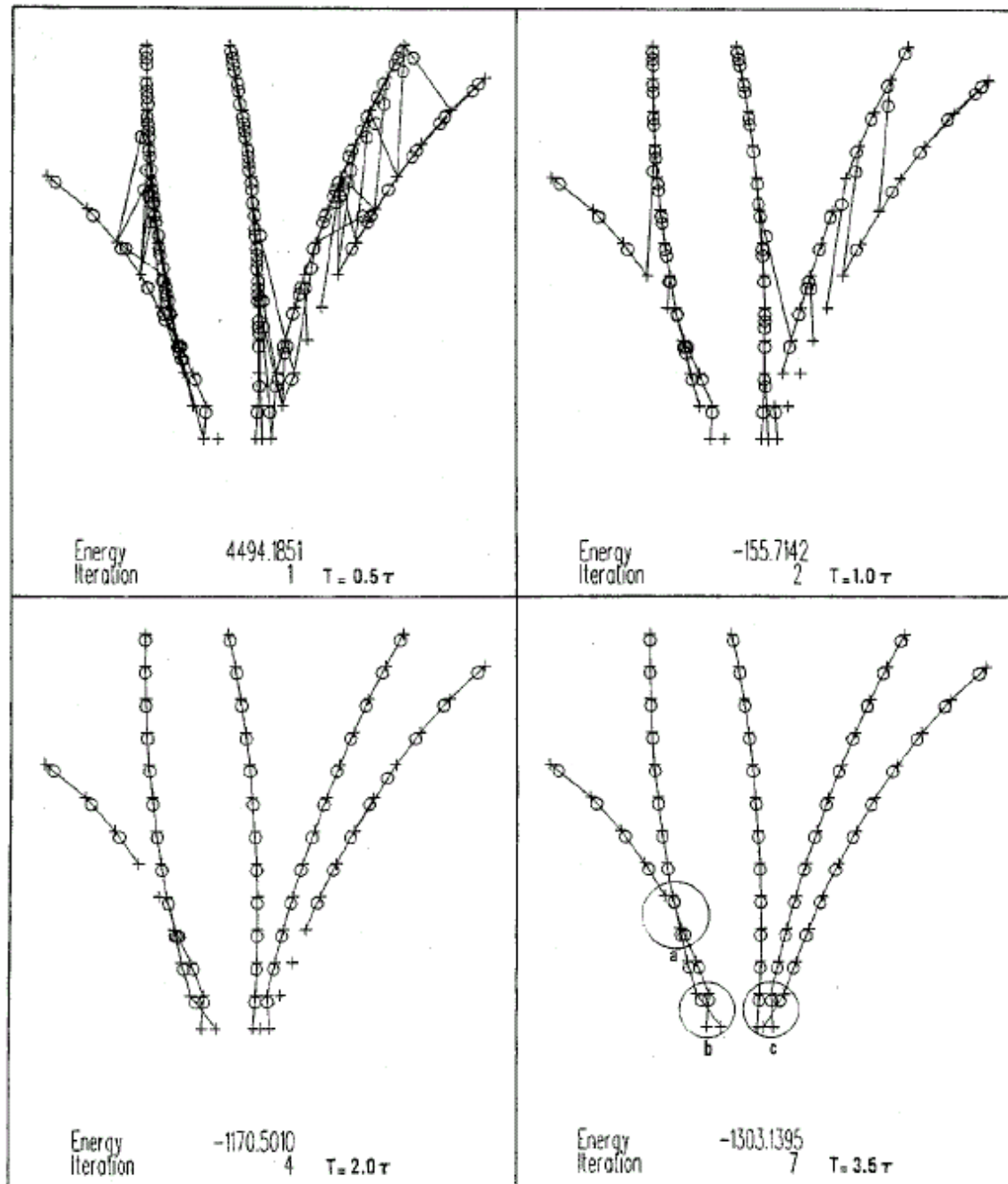
Denby-Peterson neural net

- with discrete states, minimization not very stable
- therefore, define continuous states and an update function

$$v_{ij} = \frac{1}{2} \left(1 + \tanh \left(-\frac{\partial E}{\partial v_{ij}} \frac{1}{T} \right) \right)$$

- where the temperature T is yet another adjustable parameter
- the algorithm now becomes
 - create neurons, initialize with some state value. usually a cut-off on d_{ij} is used to limit number of neurons
 - calculate the new states for all neurons using equation above
 - iterate until things have converged, eventually reducing temperature between iterations ('simulated annealing')

evolution of Denby-Peterson neural net



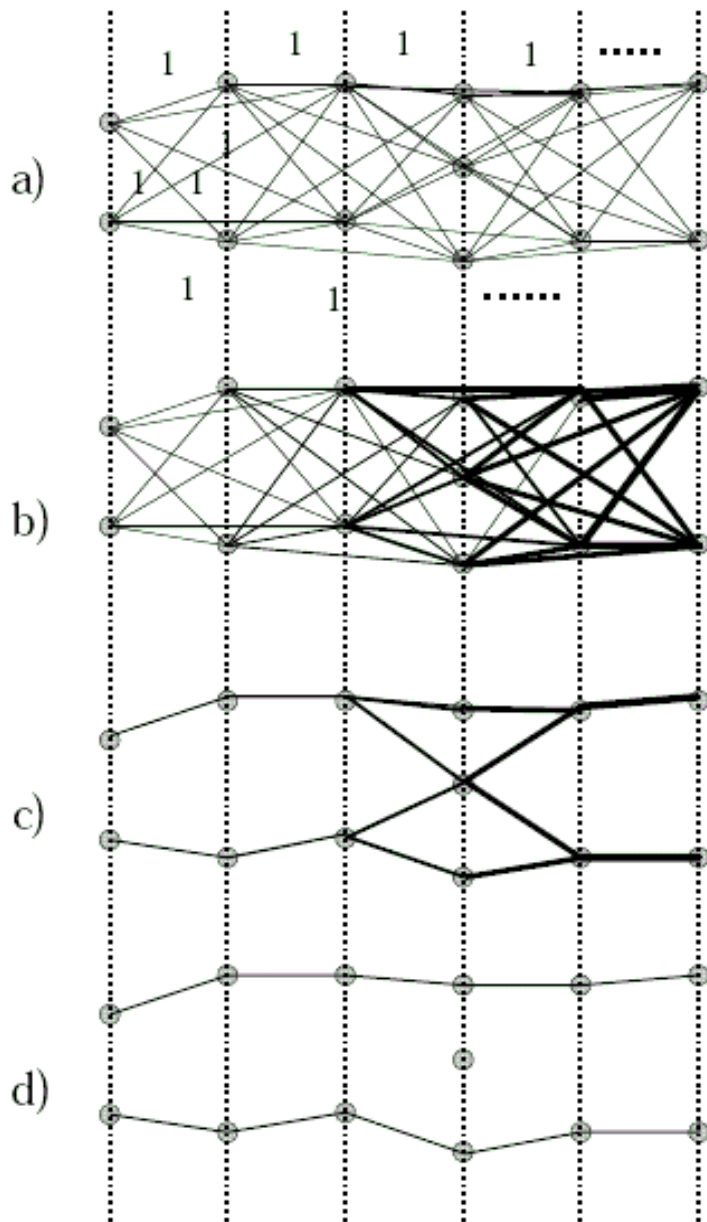
cellular automaton

- like Denby-Peterson, but simpler
- start again by creating neurons ij
 - to simplify things, connect only hits on different detector layers
 - each neuron has integer-valued state S_{ij} , initialized at 1
- make a choice about which neuron combination could belong to same track, for example, just by angle: $\theta_{kij} < \theta^{\max}$
- evolution: update all states simultaneously by looking at neighbours in layer before it

$$S_{ij} = \max\{S_{ki} | \theta_{kij} < \theta^{\max}\} + 1$$

- iterate until all cells stable
- select tracks by starting at highest state value in the network

illustration of 'CATS' algorithm



initialization

end of evolution:
state value indicated by line thickness

selection of longest tracks

more selection to remove
overlapping tracks with same
length

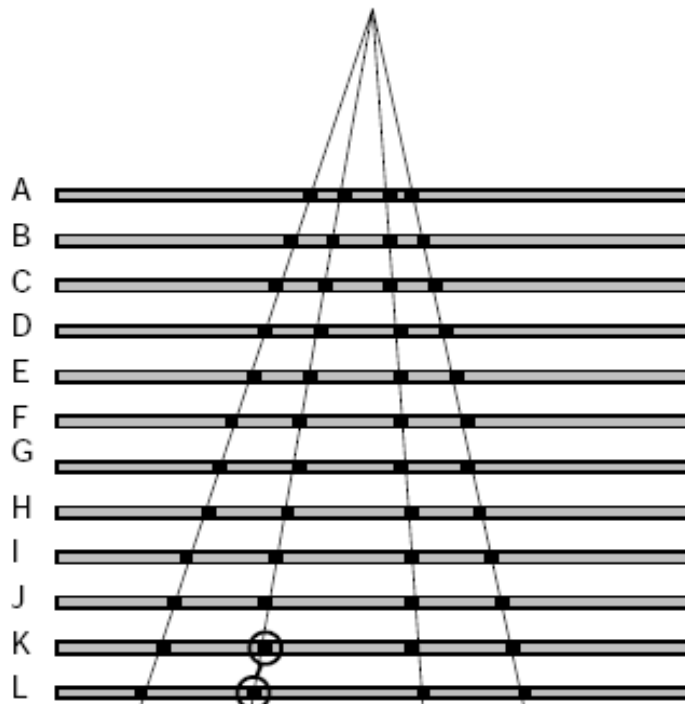
elastic arms

- ANN techniques that we just discussed
 - work only with hits in 2D or space points in 3D
 - are entirely oblivious to track model
 - just finds something straight : no difference between track bended in magnetic field and track with random scatterings
 - hard to extend to situation with magnetic field
- limitations are (somewhat) overcome by the **elastic arms algorithm**, which works with **deformable track templates**
 - neurons connect hits to finite sample of track 'templates'
 - number of templates must roughly correspond to expected multiplicity
 - main problem is sensible initialization of template parameters
 - too much for today: if you are interested, look in the literature

seed construction for local methods

- local or track following methods find tracks by extrapolating seed
- usually, seeds are created in region with lowest occupancy
- two different methods of seed construction:

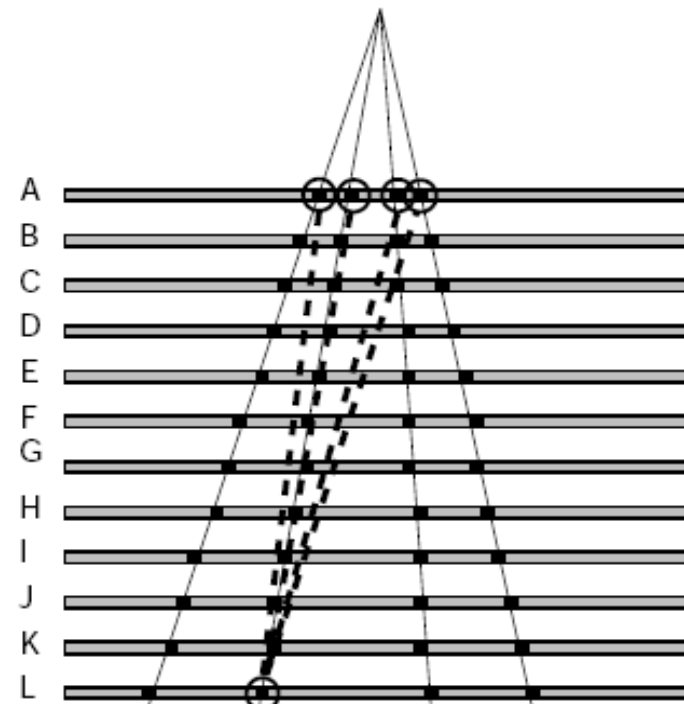
'nearby layer' approach



smaller combinatorics

worse seed parameters

'distant layer' approach



larger combinatorics

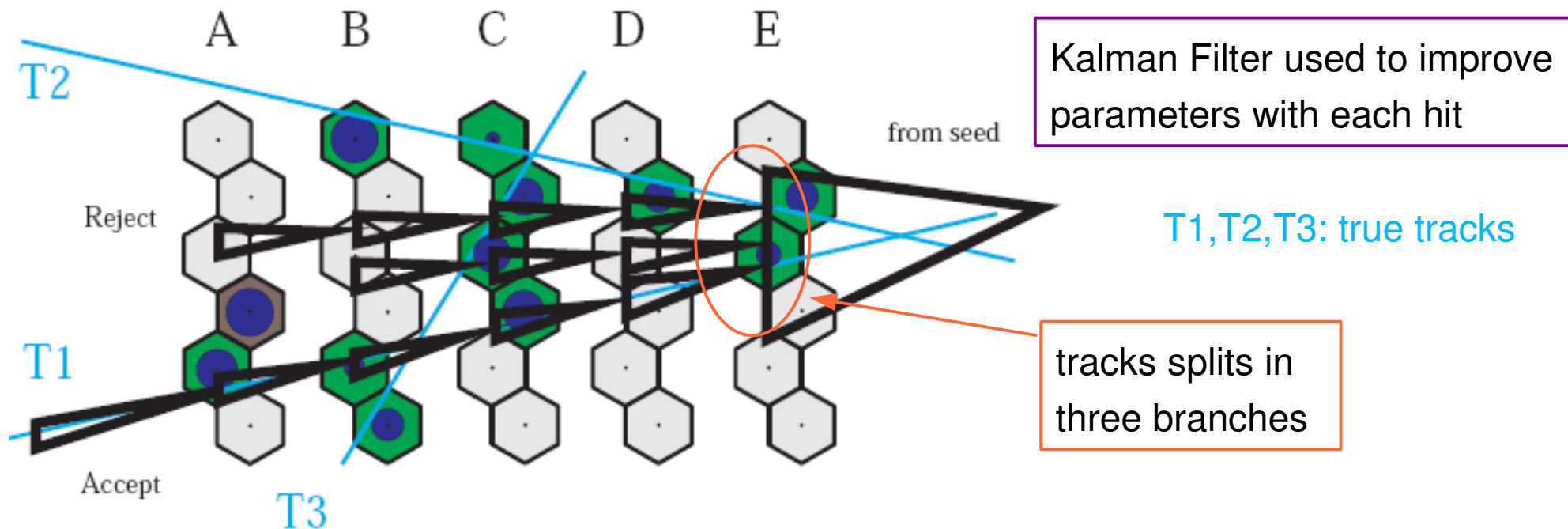
better seed parameters

track following

- track following works both in 2D and in 3D
- most simple scenario
 - navigate track candidate to next layer
 - pick closest hit within certain fixed window
 - reject track if hit is missing
- problems with this 'naïve' scenario
 - detector inefficiency may lead to track being rejected for wrong reason
 - wrong hit may be closer than correct hit
 - left-right ambiguity can not always be resolved --> may make wrong choice and spoil track

combinatorial track following

- combinatorial track following uses **candidate branching**
 - split seed if more than one hit compatible
 - follow both seeds, reject seeds with too many missing hits
 - after all layers processed, select between overlapping tracks
 - figure of merit: number of hits/holes, track chi-square etc.
- example: RANGER algorithm used in Hera-B (until replaced by CATS)



some concluding remarks

- track finding strategies are not independent of detector design
 - think how you will find tracks before building your detector
- strategies developed on MC usually need retuning once there is data
 - noise, efficiency, occupancy
- most robust strategies involve more than one track finding algorithm
 - find tracks in system A, extrapolate to B
 - find tracks in B, extrapolate to A
 - use seeds from trigger
 - etc
- there is no one-size-fits-all