UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

A multi threaded global tracking algorithm based on directed acyclic graphs and machine learning

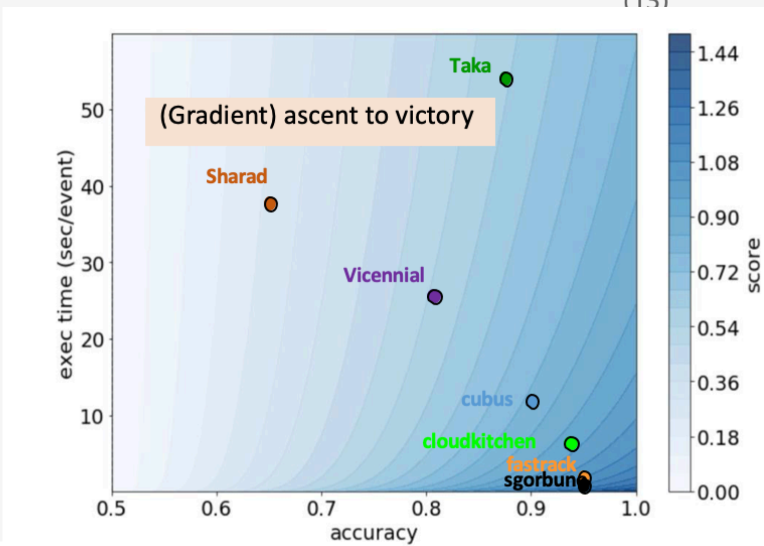
Marcel Kunze, Heidelberg University

Throughput phase Leader Board

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

RESULTS

#	User	Entries	Date of Last Entry	score ▲	accuracy_mean ▲	accuracy_std ▲	computation time (sec) ▲	computation speed (sec/event) ▲	Duration ▲
1	**sgorbuno**	9	03/12/19	1.1727 (1)	0.944 (2)	0.00 (14)	28.06 (1)	0.56 (1)	64.00 (1)
2	**fastrack**	53	03/12/19	1.1145 (2)	0.944 (1)	0.00 (15)	55.51 (16)	1.11 (16)	91.00 (6)
3	**cloudkitchen**	73	03/12/19	0.9007 (3)	0.928 (3)	0.00 (13)	364.00 (18)	7.28 (18)	407.00 (8)
4	cubus	8	09/13/18	0.7719 (4)	0.895 (4)	0.01 (9)	675.35 (19)	13.51 (19)	724.00 (9)
5	Taka	11	01/13/19	0.5930 (5)	0.875 (5)	0.01 (12)	2668.50 (23)	53.37 (23)	2758.00 (13)
6	Vicennial	27	02/24/19	0.5634 (6)	0.815 (6)	0.01 (10)			
7	Sharad	57	03/10/19	0.2918 (7)	0.674 (7)	0.02 (4)			
8	WeizmannAI	5	03/12/19	0.0000 (8)	0.133 (11)	0.01 (11)			
9	harshakoundinya	2	03/12/19	0.0000 (8)	0.085 (13)	0.01 (6)			
10	iWit	6	03/10/19	0.0000 (8)	0.082 (15)	0.01 (8)			
				0.0000					



Throughput phase 3rd place

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Phase 2 cloudkitchen

Author: Marcel Kunze

ROOT 6 C++ CPP

Accuracy: 0.93
Time/event: ~7 sec
Memory: 0.7 Gb

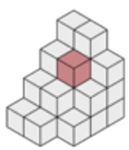
partly based on top quarks Phase 1 solution

1	—	**Top Quarks**	0.92182	10	2mo

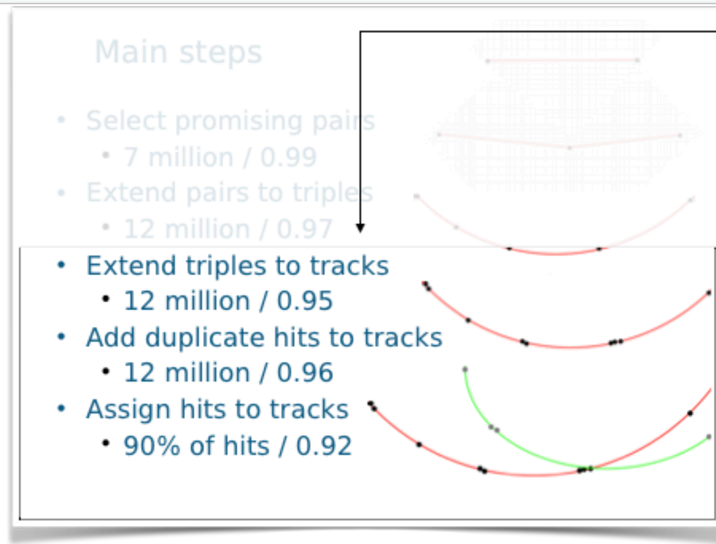
Algorithm outline

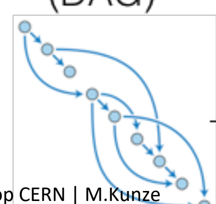
hits

sorted in voxels



organised in
direct acyclic graphs
(DAG)





DAGs are pre-trained on ~25 events ground truth

DAGs are used to fast navigate through voxel space

Disc section

Tube section

Threaded

doublet finder

NN1 NN2

Triplet finder

NN3

ca. 300k
97.2%

ca. 500k
99.4%

ca. 2 Mio.

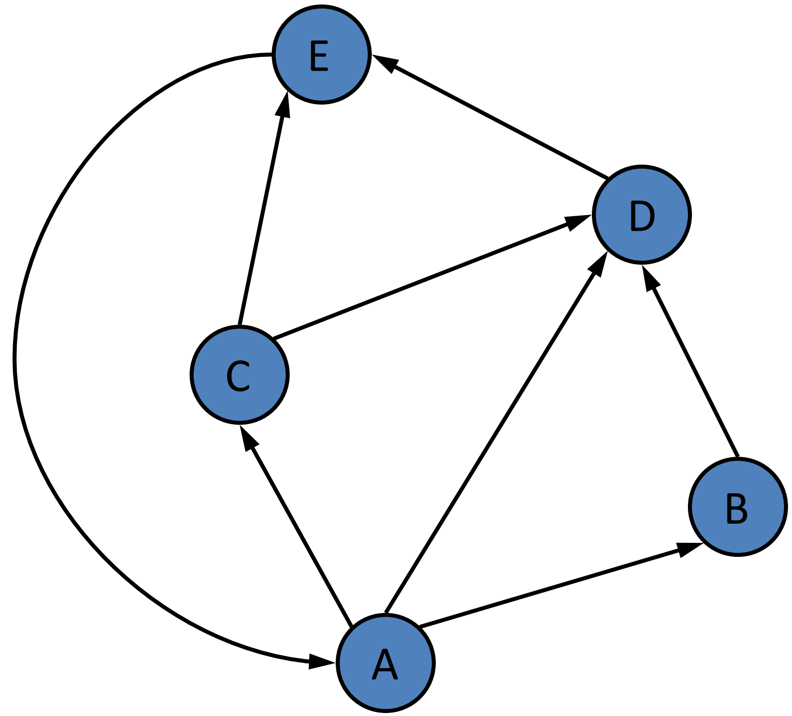
Directed Graphs

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

A directed Graph is a graph whose edges are all directed

Applications

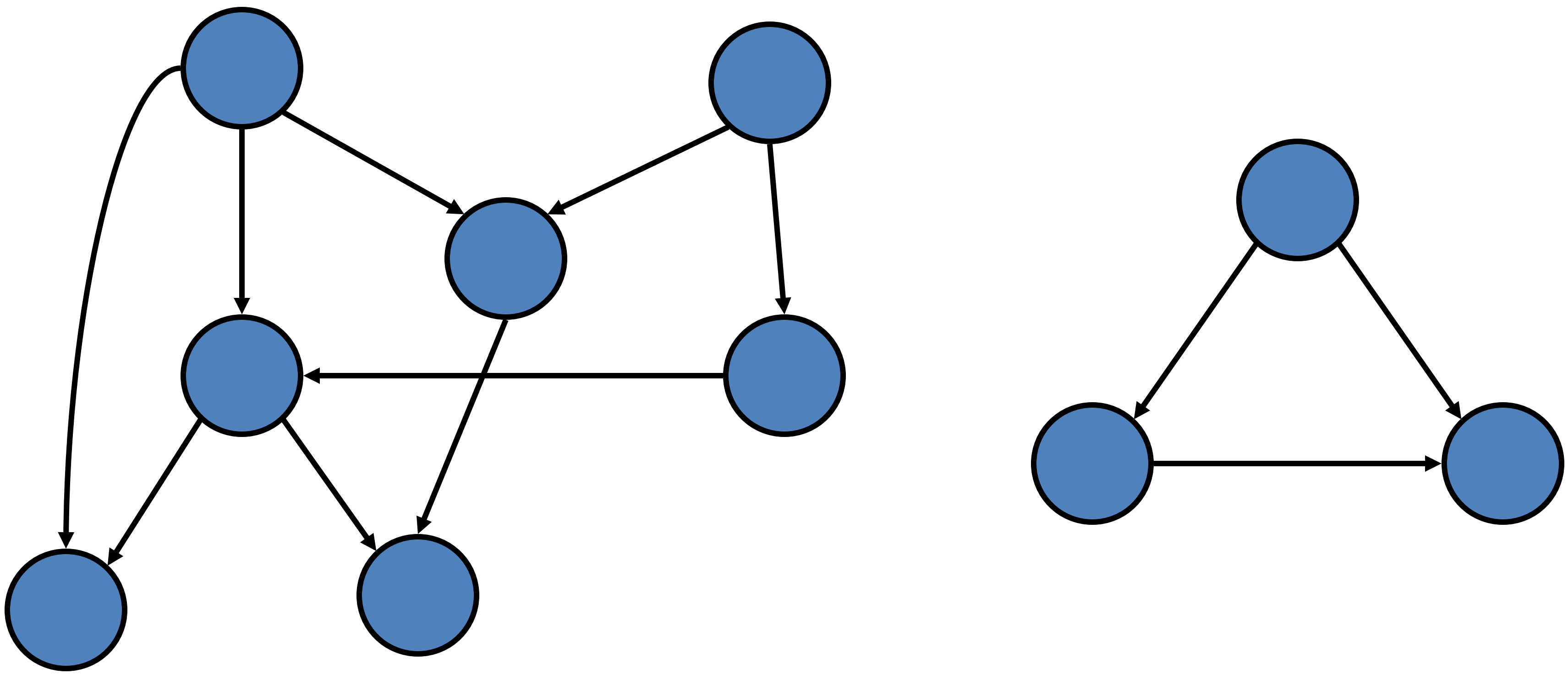
- one-way streets
- flights
- task scheduling
- …



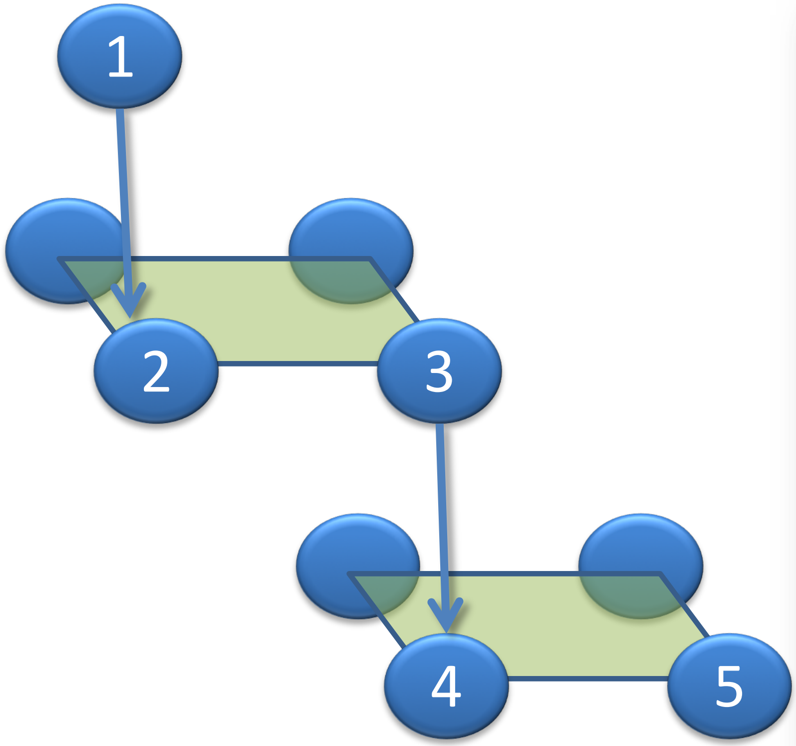
Directed Acyclic Graphs (DAG)

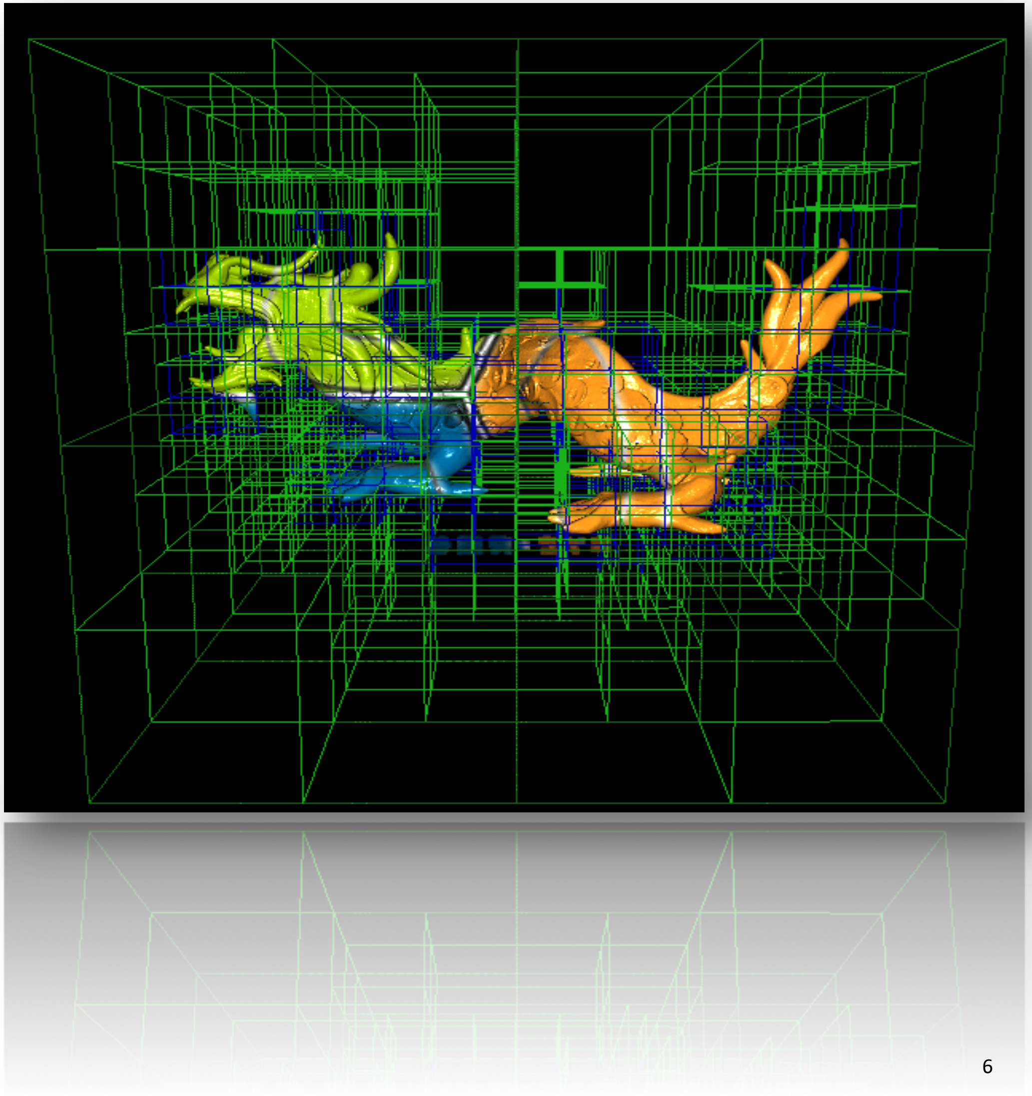
UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

A *directed acyclic graph* or *DAG* is a directed graph with no directed cycles:



Gaming: Sparse Voxel Octrees (SVO)





- Raytracing
- Compression of data
- Multi-scale resolution

Voxel (Volume Pixel)

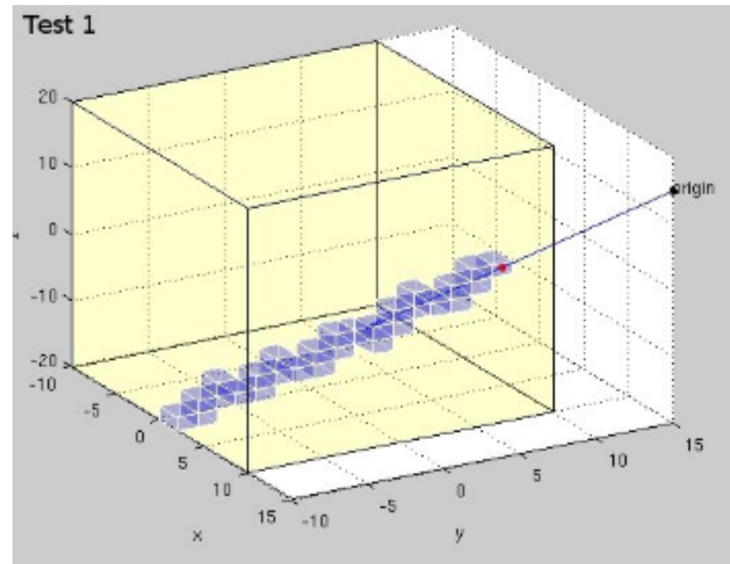
UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Define spatial elements in $\phi * \theta$ (voxel)

- Organize the voxels in DAGs according to track evolution in radial direction

index = (phi<<32) | (theta<<24) | (layer<<16) | module;

- Flexible to model even arbitrary paths (kinks, missing hits, outliers, random walk, ..)
- Training is done with MC tracks of typically 15-25 events

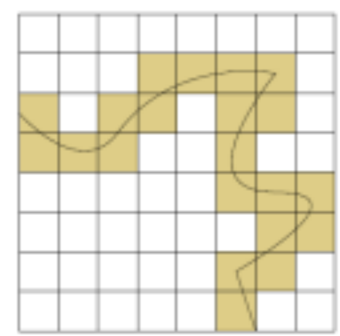


Multiscale resolution (Better use SVOs?)

- 2*1 DAGs for pair finding (slices)
- 12*14 DAGs for triple finding (tiles)

Path finding

- Sort event hits into the trained DAGs
- Seed and follow the path strategy



Pattern Recognition with Machine Learning

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Intuition
- Model free estimator
- Start with basic quantities
- Coordinates, simple derived values
- Only very basic detector specific information

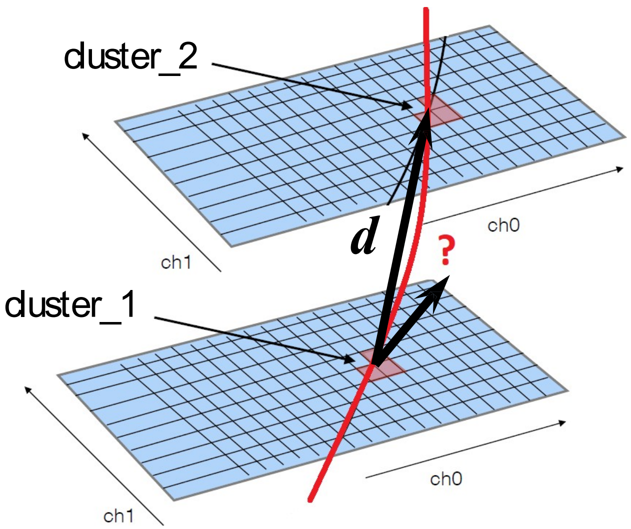
Input parameter space
- Polar coordinates (R_t, ϕ, z)
- Directional cosines
- Simple helix calculation (score)

In principal not needed, but speeds up the thing !

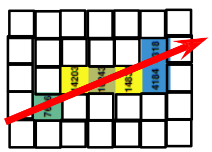
Training
- Supervised: presenting MC ground truth
- Unsupervised: presenting probability density function

Input Parameter Space

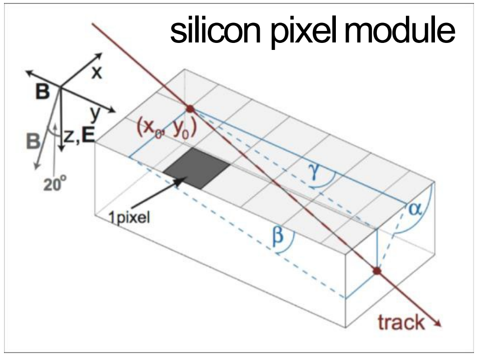
Given two hits (clusters of silicon cells): predict if they belong to the same track



- Estimate track direction from the cluster shape:



eigenvector of covariance matrix of the silicon cells



Features for the training

- Polar coordinates of the hit doublet: (r_1,ϕ_1,z_1), (r_2,ϕ_2,z_2)
- Triplet finder works the same with a hit triplet
- Simple helix score
- Angle/length deviations of the vector ***d*** projection from the values predicted by the shape of cluster 1
- Angle/length deviations of the vector ***d*** projection from the values predicted by the shape of cluster 2

Input Parameter Folding

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

The tracking problem is symmetric wrt. polar coordinates

- Fold the input parameter space into an octagon slice using “abs” function
- Considerable improvement of the separation strength of the parameters
- Need less statistics / yield better results

Rank	Variable	Separation
1	log(score)	5.039e-01
2	rz3	5.491e-04
3	phi3	7.552e-05
4	z3	4.986e-05
5	rz2	1.519e-05
6	rz1	9.568e-06
7	phi2	4.101e-06
8	z1	1.967e-06
9	z2	1.965e-06
10	phi1	1.503e-06

Rank	Variable	Separation
1	log(score)	5.978e-01
2	rz3	6.329e-04
3	abs(abs(phi3)-1.57079632679)	1.317e-04
4	abs(z3)	5.522e-05
5	rz2	2.067e-05
6	rz1	1.675e-05
7	abs(abs(phi2)-1.57079632679)	4.335e-06
8	abs(z1)	3.592e-06
9	abs(abs(phi1)-1.57079632679)	3.038e-06
10	abs(z2)	2.963e-06

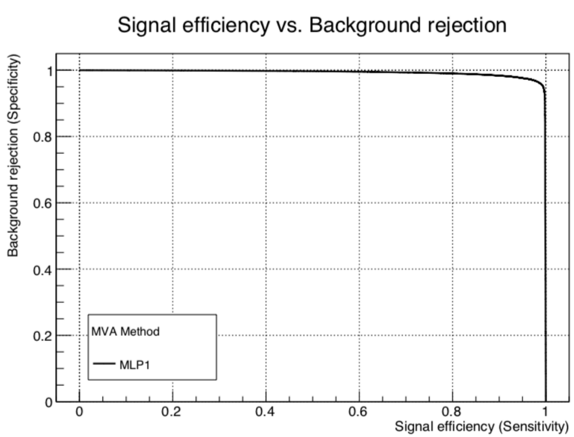
Hit Doublet / Triplet Classification: MLP

“Shallow learning” ;)

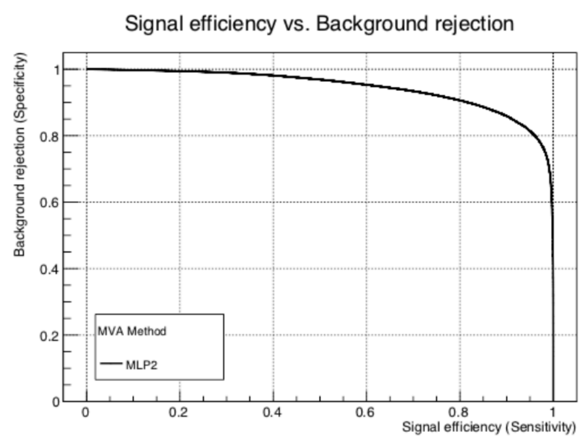
UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- **Classify the doublets and triplets with neural networks**
  - Multi Layer Perceptron: MLP1 8-15-5-1 / MLP2 9-15-5-1 / MLP3 10-15-5-1
  - Input: hit coordinates, directional cosines towards the clusters, helicity score wrt. origin
  - Output: doublet/triplet quality, supervised training with Monte-Carlo ground truth
  - Training: Typically 10 events, O(Mio) patterns, 500 epochs, one hour on standard PC
  - “*Receiver Operation Characteristics*” (ROC) curves indicate good quality

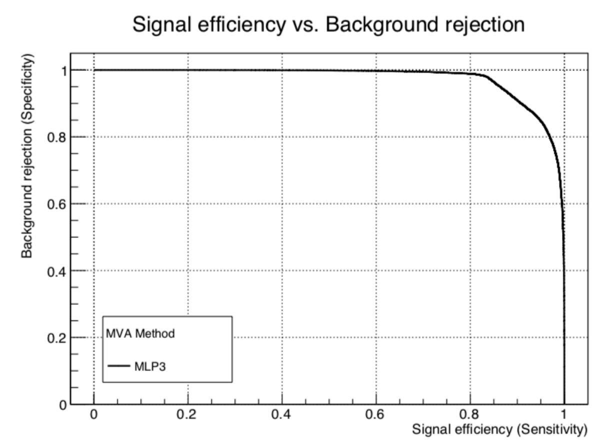
Doublet finder (disc)



Doublet finder (tube)



Triplet finder



Worse due to vertex shift !

Hyperparameter Tuning

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Automated tests with docker / singularity to maximize CodaLab score
Test set of 50 events not used by training. Optimize:

- Spatial resolution / training of DAGs
- Network topology and cuts on output wrt. event size
- Run time / accuracy trade-offs

```
Timer0 initHits 12.000000 ms
Timer1 initCells 0.000000 ms
Timer2 initGraphData 27.000000 ms
Timer3 initHitDir 11.000000 ms
Timer4 initPolarModule 202.000000 ms
Timer5 initRecoObjects 0.000000 ms
Timer6 initTasks 0.000000 ms
Timer7 findCandidatesGraph 1136.000000 ms
Timer8 findTriplesGraph 1967.000000 ms
Timer9 findPaths 816.000000 ms
Timer10 addDuplicates 762.000000 ms
Timer11 findAssignment1 774.000000 ms
Timer12 findAssignment2 106.000000 ms
Timer13 mapAssignment 29.000000 ms
Timer14 writeSubmission 10.000000 ms
Processing time per event  5852.000000 ms
```

Files 20, Phi / Theta	8	10	12	14	16
8	0.883360		0.878024		
10		0.880177	0.884479	0.887572	0.885034
12	0.878399	0.883600	0.887683	**0.889858**	0.881736
14		0.880297	0.877356	0.884148	0.878094
16		0.882559	0.885102	0.876590	0.871375

T3 / hits	90000	100000	110000	120000	130000	140000
0.2	0.896278					
0.3	0.896026					
0.4	**0.896748**	0.871153	0.847126			
0.5	0.895815	**0.871703**	**0.847288**	0.825986	0.806712	0.779419
0.6	0.893367	0.871531	0.847247	**0.826128**	**0.806855**	0.780699
0.7			0.846020	0.825648	0.806230	**0.780974**
0.8						0.779338

Multi Threading

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- **Well defined algorithmic steps for pattern recognition**
- **Efficient parallelism on the basis of DAGs**
  - Form doublets from seeding hits in a DAG (MLP1, MLP2)
  - Extend the doublets to triplets (MLP3)
  - Extend the triplets to path segments
  - The path segments are merged into tracklets
  - Remove duplicate solutions

The tracklets are merged into a common tracking solution by **serial tasks**

```
Timer0 initHits 12.000000 ms
Timer1 initCells 0.000000 ms
Timer2 initGraphData 27.000000 ms
Timer3 initHitDir 11.000000 ms
Timer4 initPolarModule 202.000000 ms
Timer5 initRecoObjects 0.000000 ms
Timer6 initTasks 0.000000 ms
Timer7 findCandidatesGraph 1136.000000 ms
Timer8 findTriplesGraph 1967.000000 ms
Timer9 findPaths 816.000000 ms
Timer10 addDuplicates 762.000000 ms
Timer11 findAssignment1 774.000000 ms
Timer12 findAssignment2 106.000000 ms
Timer13 mapAssignment 29.000000 ms
Timer14 writeSubmission 10.000000 ms
Processing time per event  5852.000000 ms
```

Serial tasks: ca. 0.3 seconds

Parallel tasks: ca. 4 seconds

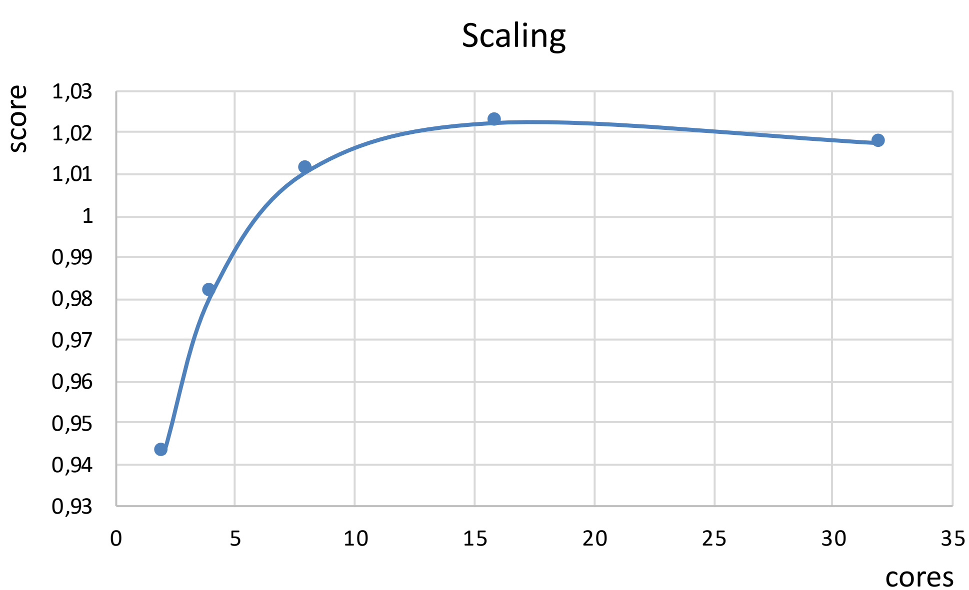
Serial tasks: ca. 0.8 seconds

Scaling Behavior

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Scaling tests have been performed with Amazon EC2

- Instance type c5n.9xlarge (36 cores)
- Core power comparable to CodaLab cores
- Code scales up to 16 cores (Score: 1.022, accuracy 92.3%, 1.7s)
- Limited by serial code: Sorting tracklets into tracks (improve by use of OpenMP ?)



Amdahls Law: Speedup is the fraction of code P that can be parallelized:

$$speedup = \frac{1}{1-P}$$

Machine Learning Advantage

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Model free estimator

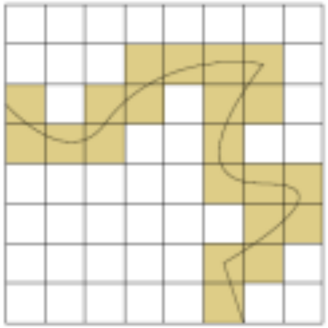
- Solution may be easily transferred to a different context

Graceful degradation in presence of changes

- Geometry
- Dead channels
- Calibration
- …

The DAGs may represent arbitrary tracking paths

- Inhomogeneous magnetic field
- Kinks
- …



Machine Learning Software: Neural Network Objects

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Neural Network Objects (NNO) is a C++ class library for Machine Learning based on the ROOT framework

Supervised models
- Multi-Layer Perceptron (TMLP, TXMLP)
- Fisher Discriminant (TFD)
- **Supervised Growing Cell Structure (TSGCS)**
- **Supervised Growing Neural Gas (TSGNG)**
- Neural Network Kernel (TNNK)

Unsupervised models
- Learning Vector Quantization (TLVQ)
- **Growing Cell Structure (TGCS)**
- **Growing Neural Gas (TGNG)**

Published on https://github.com/marcelkunze/rhonno

The solution has also been trained with ROOT/TMVA, yields comparable results.

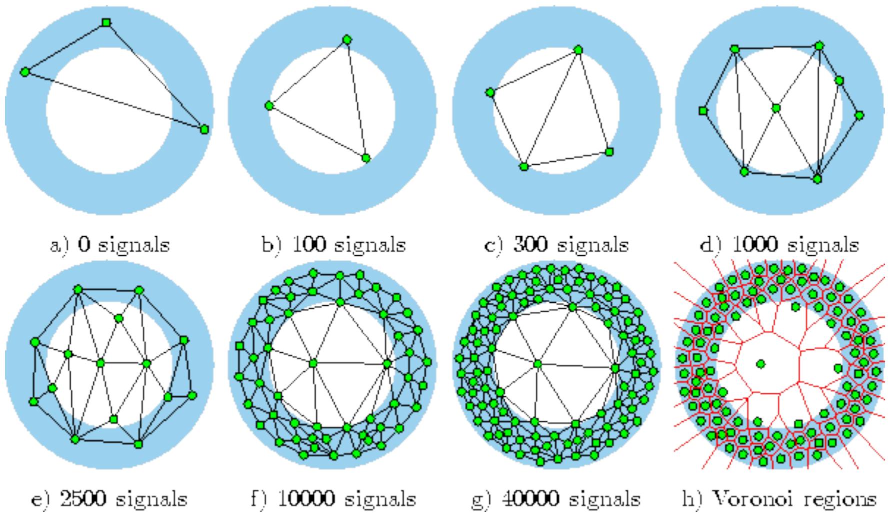
Unsupervised Learning

By use of **unsupervised training** we may

- fit probability density functions without a-priori knowledge of MC-Truth
- identify the tracks according to their measured intrinsic properties

Self organizing neural networks

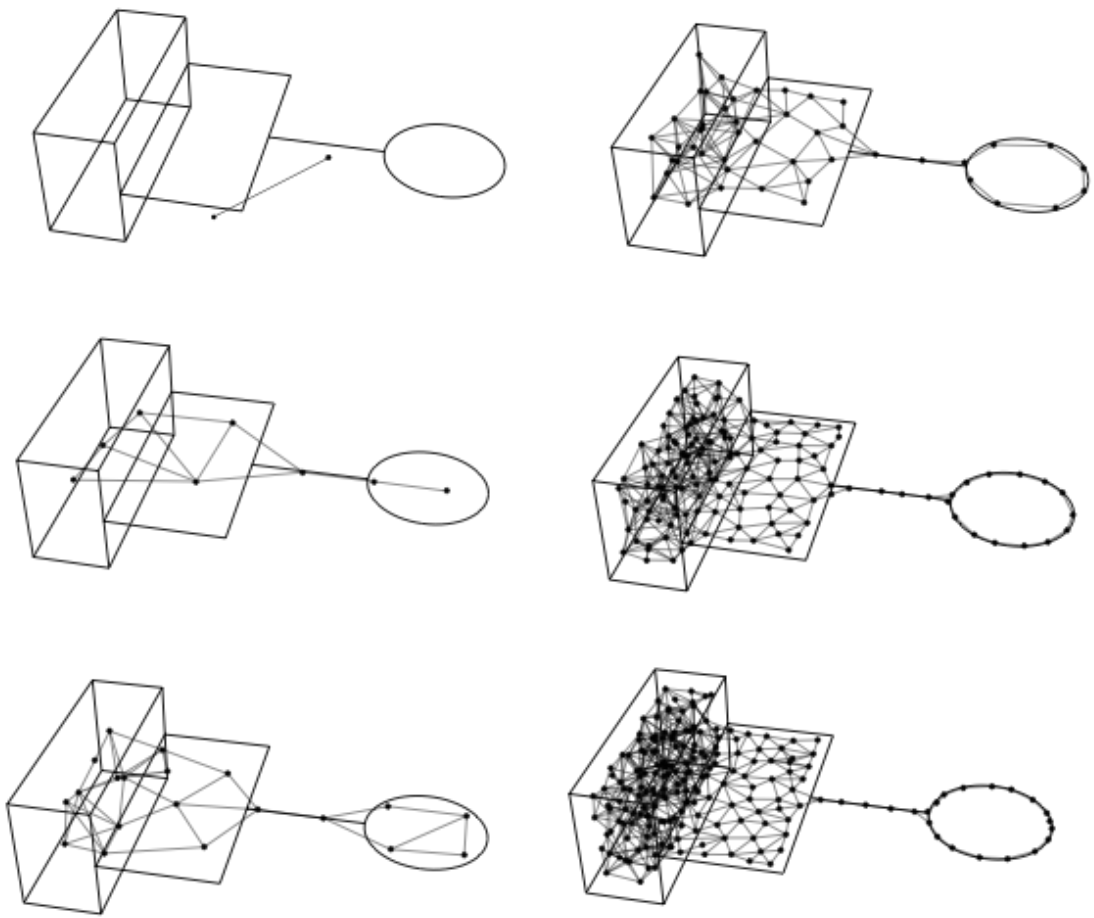
e.g. Growing Cell Structures (GCS)



Example:
GCS learns ring-shaped probability density function

Growing Neural Gas (GNG) expands the model to mixed Dimensions

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386



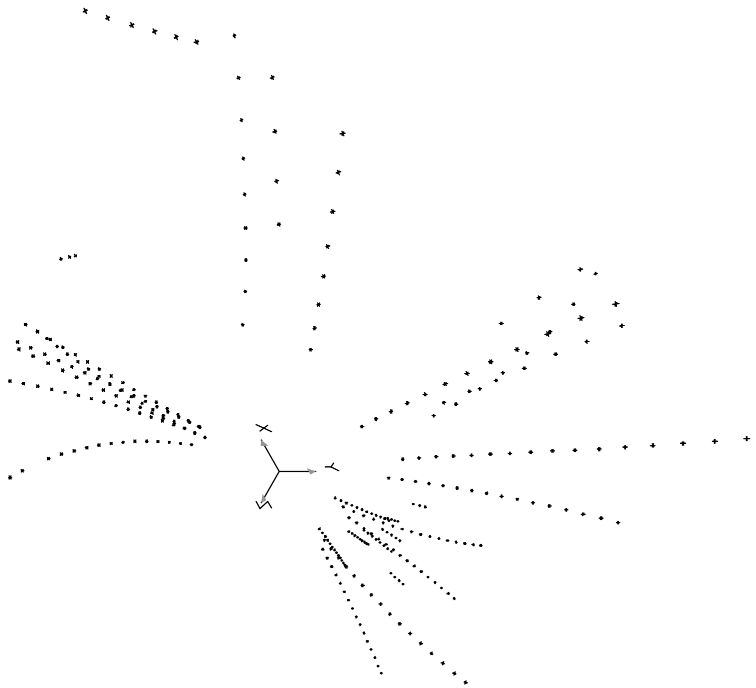
M.Kunze: Growing Cell Structure and Neural Gas - Incremental Neural Networks, New computing techniques in physics research IV (Ed. B. Denby), World Sci., 1995

Tracking Example

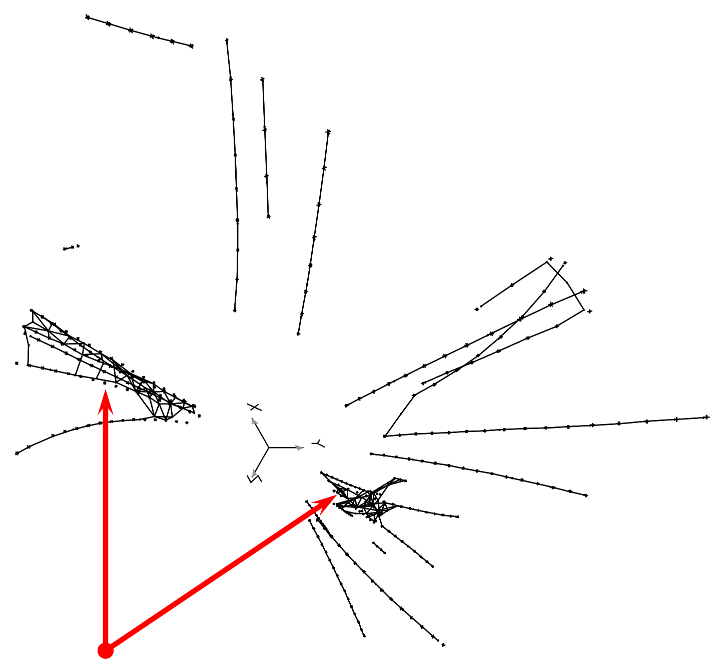
UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Intuition: Use a GNG self organizing neural network to grow along particle trajectories

- Move and connect nodes according to the density of measured points



Example: 22 Tracks, 309 Points
Measured by DELPHI TPC (1989)

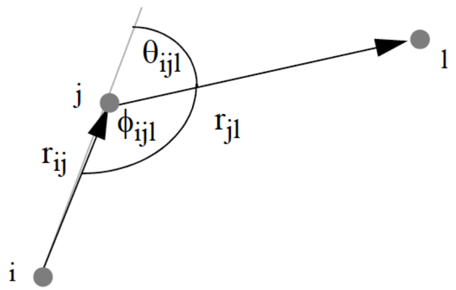


Convergence of training after 60 epochs
Problem: crossover connections between tracks in regions of high track density

How to get straight tracks?

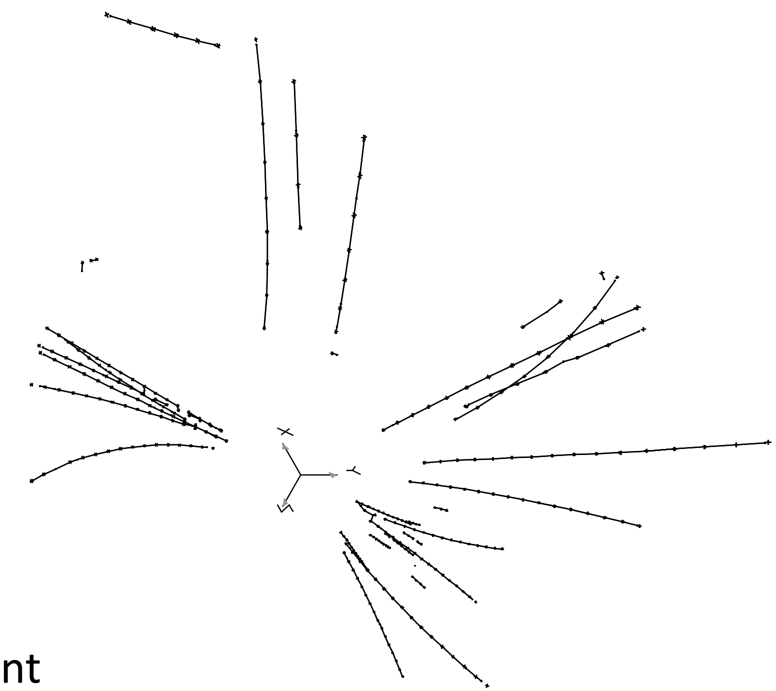
A) Apply classical cuts

- insert new nodes at the neighboring measured point
- introduce a „rotor model“ to support the track representation
- remove connections with large angle to suppress crossover connections: require $|\cos\Phi| < \varepsilon$



B) Check GNG triplets with MLP

- Re-use MLP3 from trackML contest (!!!)
- Checking of triplets works astonishingly well

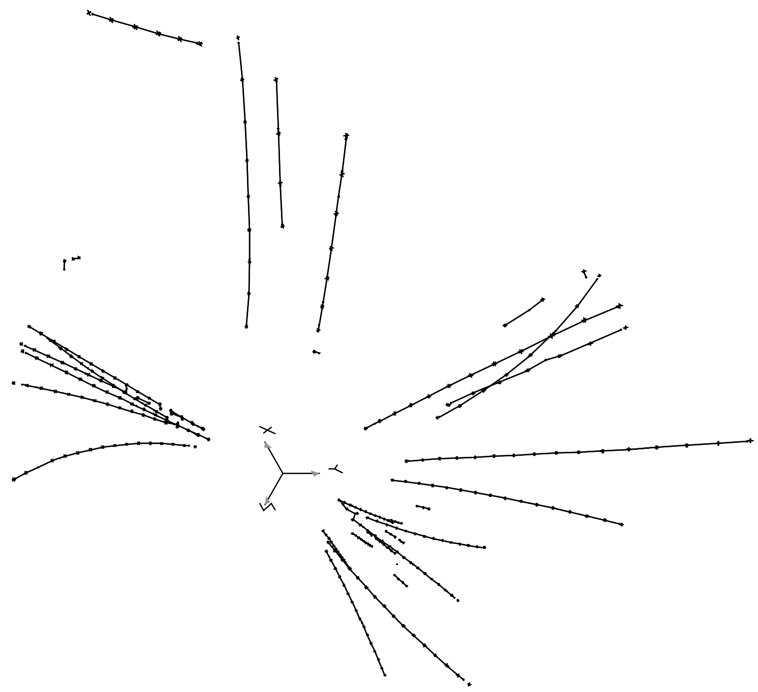


Challenges

- Difficult to handle high track density
- Compute intense, training happens for each event

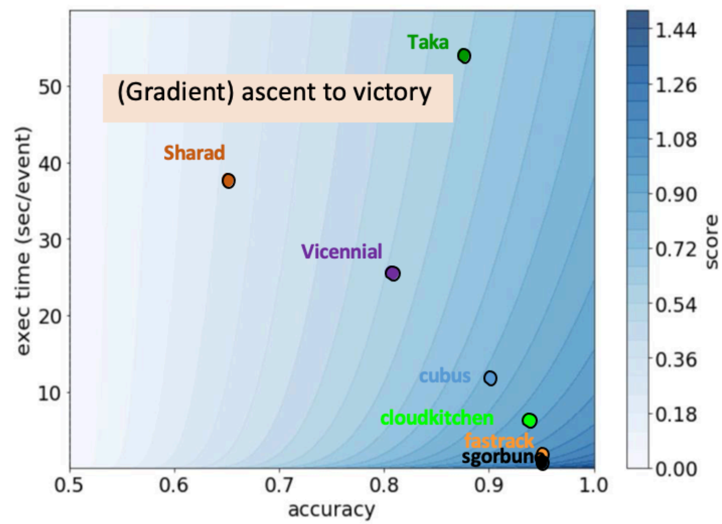
Live Demonstration

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386



Summary

- ML works well with tracking problems
- Supervised training of MLPs for pattern recognition can be fast and accurate
- Unsupervised training of incremental networks is interesting as it is completely data driven (No need for MC truth / model free estimator)



Dr. Marcel Kunze

marcel.kunze@uni-heidelberg.de

Im Neuenheimer Feld 293 / 106

D-69120 Heidelberg

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386