# Ideas for VELO tracking

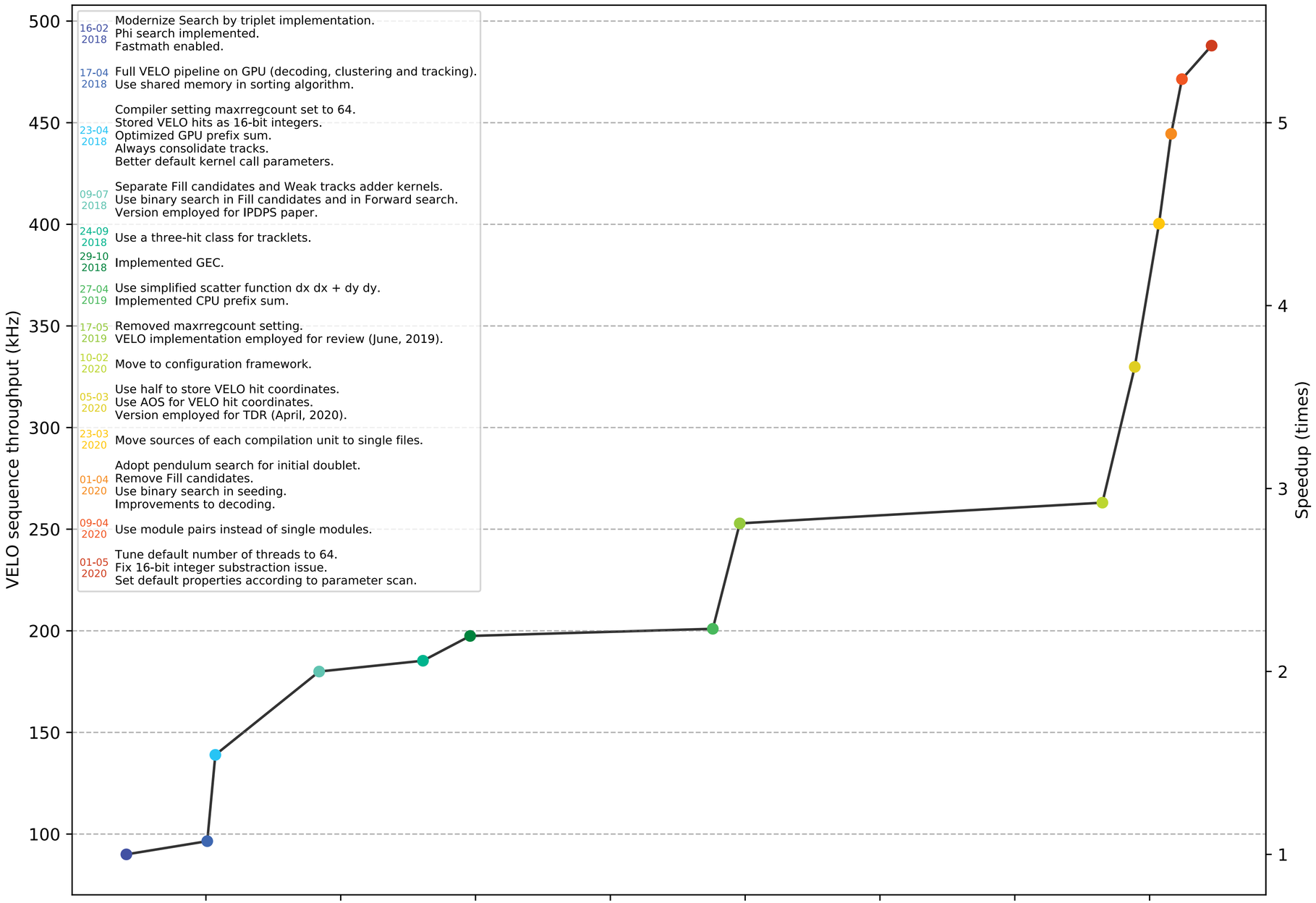
Daniel Cámpora

Maastricht University

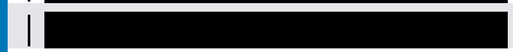
Department of Data Science and
Knowledge Engineering

# State of the art

- Velo reconstruction is one of the most relevant track reconstruction problems in LHCb. Especially in the real-time trigger.
    - As the LHCb detector evolves, the problem keeps becoming harder. **Scalability is key**.
- It impacts the efficiency of the rest of the tracking algorithms and the primary vertex finder.
- We have optimized algorithms to solve it over many iterations.


VELO Search by triplet performance evolution (GeForce RTX 2080 Ti)
16-02 2018 Modernize Search by triplet implementation.
Phi search implemented.
Fastmath enabled.
17-04 2018 Full VELO pipeline on GPU (decoding, clustering and tracking).
Use shared memory in sorting algorithm.
23-04 2018 Compiler setting maxrregcount set to 64.
Stored VELO hits as 16-bit integers.
Optimized GPU prefix sum.
Always consolidate tracks.
Better default kernel call parameters.
09-07 2018 Separate Fill candidates and Weak tracks adder kernels.
Use binary search in Fill candidates and in Forward search.
Version employed for IPDPS paper.
24-09 2018 Use a three-hit class for tracklets.
29-10 2018 Implemented GEC.
27-04 2019 Use simplified scatter function dx dx + dy dy.
Implemented CPU prefix sum.
17-05 2019 Removed maxrregcount setting.
VELO implementation employed for review (June, 2019).
10-02 2020 Move to configuration framework.
05-03 2020 Use half to store VELO hit coordinates.
Use AOS for VELO hit coordinates.
Version employed for TDR (April, 2020).
23-03 2020 Move sources of each compilation unit to single files.
01-04 2020 Adopt pendulum search for initial doublet.
Remove Fill candidates.
Use binary search in seeding.
Improvements to decoding.
09-04 2020 Use module pairs instead of single modules.
01-05 2020 Tune default number of threads to 64.
Fix 16-bit integer substraction issue.
Set default properties according to parameter scan.
VELO sequence throughput (kHz)
Speedup (times)
Date
100
150
200
250
300
350
400
450
500
1
2
3
4
5
04-2018
07-2018
11-2018
02-2019
05-2019
08-2019
12-2019
03-2020

# Still the most relevant

| Algorithm | Contribution | % |
|---|---|---|
| velo_estimate_input_size_kernel |  | 14.47 |
| lf_triplet_seeding |  | 12.90 |
| velo_search_by_triplet |  | 10.96 |
| velo_sort_by_phi |  | 8.35 |
| ut_find_permutation |  | 5.19 |
| velo_masked_clustering_kernel |  | 5.18 |
| ut_search_windows |  | 3.94 |
| scifi_direct_decoder_v4_kernel |  | 2.75 |
| muon_populate_hits |  | 2.72 |
| lf_extend_tracks |  | 2.61 |
| pv_beamline_multi_fitter |  | 2.60 |

# In terms of complexity

- Track reconstruction is equivalent to *partitioning hits into disjoint sets*, considering *noise* and a *region of acceptance*.

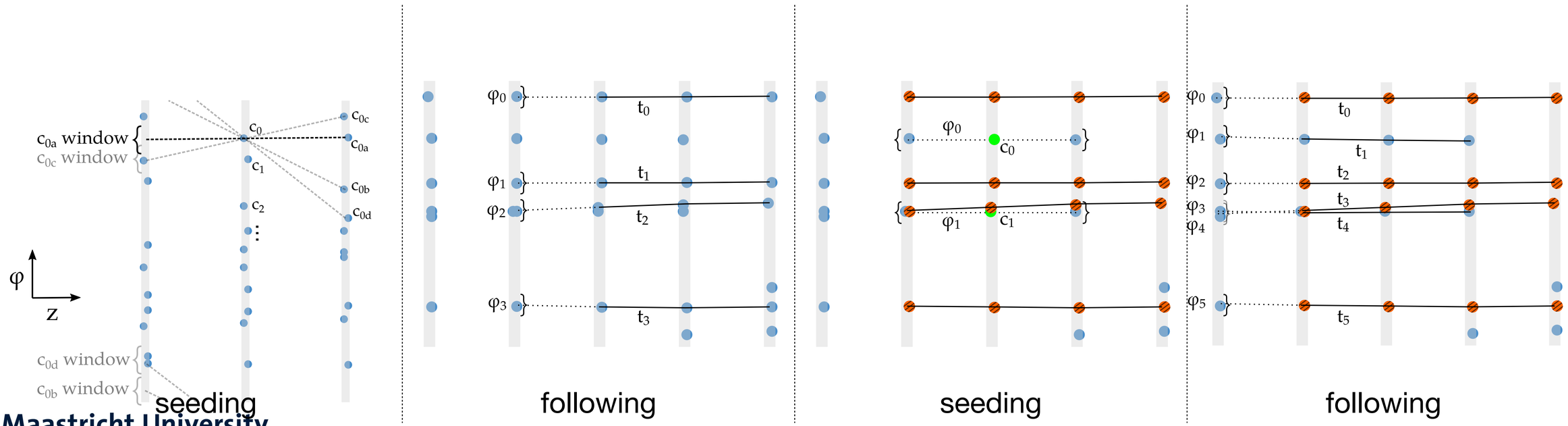
- A general solution to Velo track reconstruction with $m$ modules and $n$ hits per module is $O(n^m)$.

- When a problem is intractable, it is often better to solve a different, approximation problem instead.

# *Search by triplet*

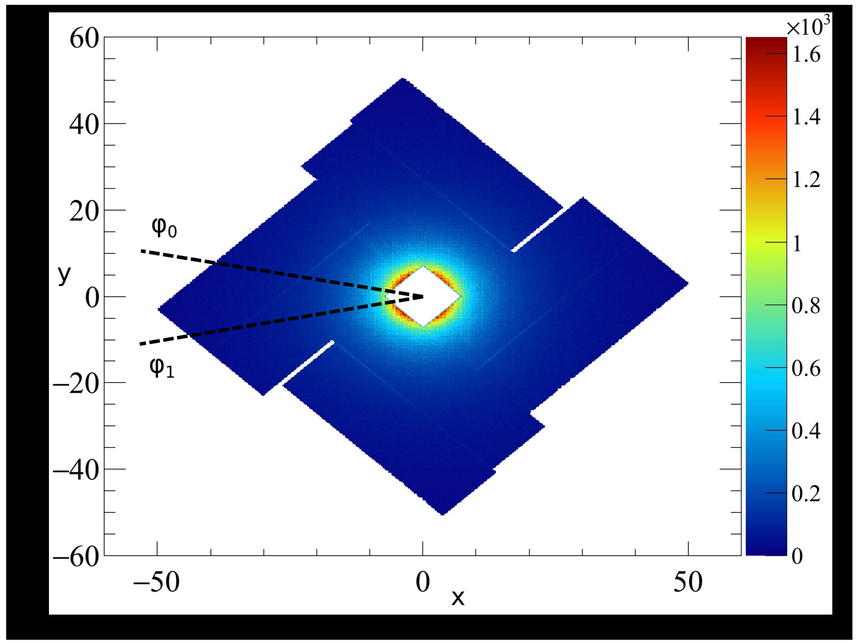
- Simplification: *Partition hits into disjoint sets, with at most one hit of the same module in each set*. Look only at a few likely candidates using binary search.

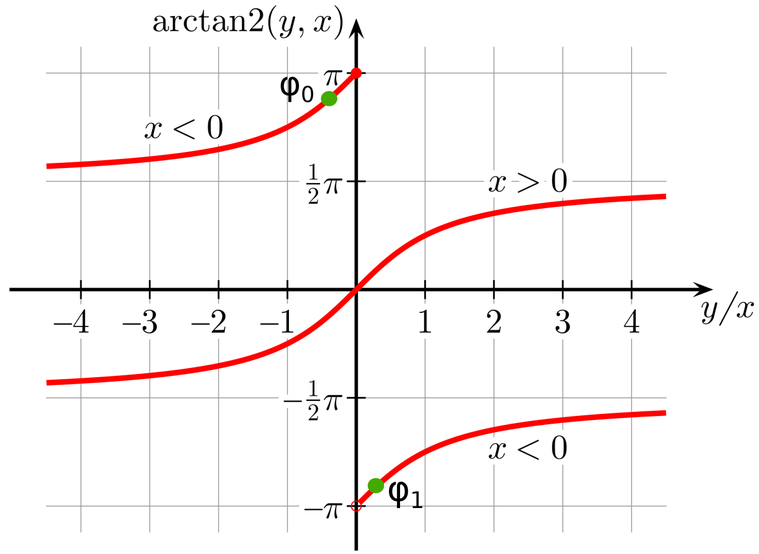
- Strategy: **Sort** hits, find **seeds** (triplets of compatible hits in neighbouring modules), **follow** tracks to remaining modules.



# Some details

- A good sorting mechanism is $\varphi = arctan(y, x)$
- We employ a circular buffer (*modulo arithmetic*).



# *Search by triplet* complexity and efficiency

- Complexity ($m$ modules, $n$ hits per module):

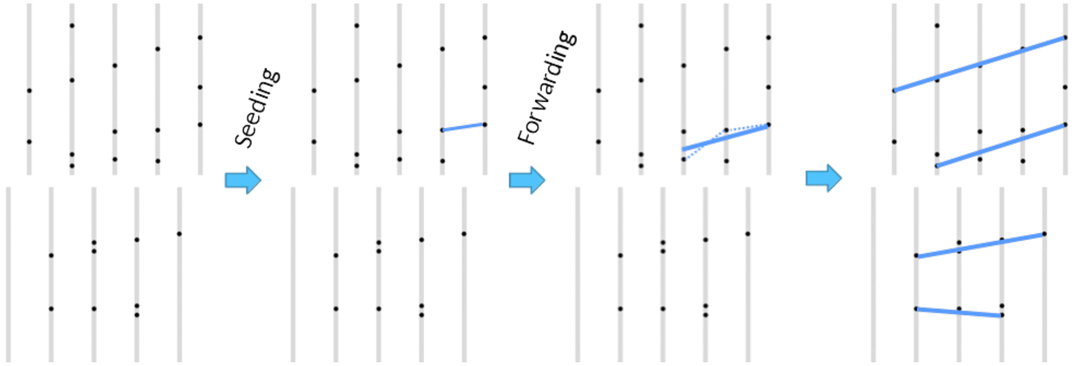
| | |
|---|---|
| Sort by phi | $O(m \cdot n \cdot log(n))$ |
| Track seeding | $O(m \cdot n \cdot log(n)^2)$ |
| Track following | $O(m \cdot n \cdot log(n))$ |
| Tracklet filter | $O(m \cdot n)$ |

- Physics efficiency:

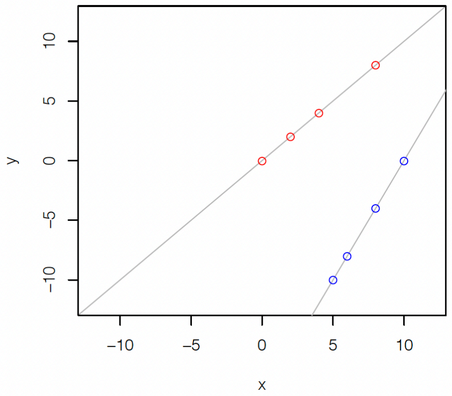
| Particle category | Reco eff. | Clone fraction | Hit purity | Hit eff. |
|---|---|---|---|---|
| All | 98.52 | 2.14 | 99.30 | 96.45 |
| Strange | 98.13 | 1.58 | 99.48 | 97.35 |
| From B | 99.30 | 1.16 | 99.74 | 98.11 |
| Electrons | 97.38 | 2.74 | 98.18 | 97.02 |
| From B electrons | 97.00 | 3.68 | 98.42 | 96.68 |
| Overall fake fraction | | 0.86 | | |

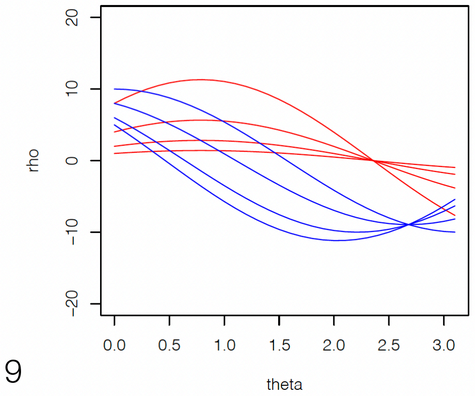
# Can we do better?

- *Local methods* - Methods that build track seeds and build on them iteratively.




- *Global methods* - Map the problem to an equivalent one, where solutions are tracks.



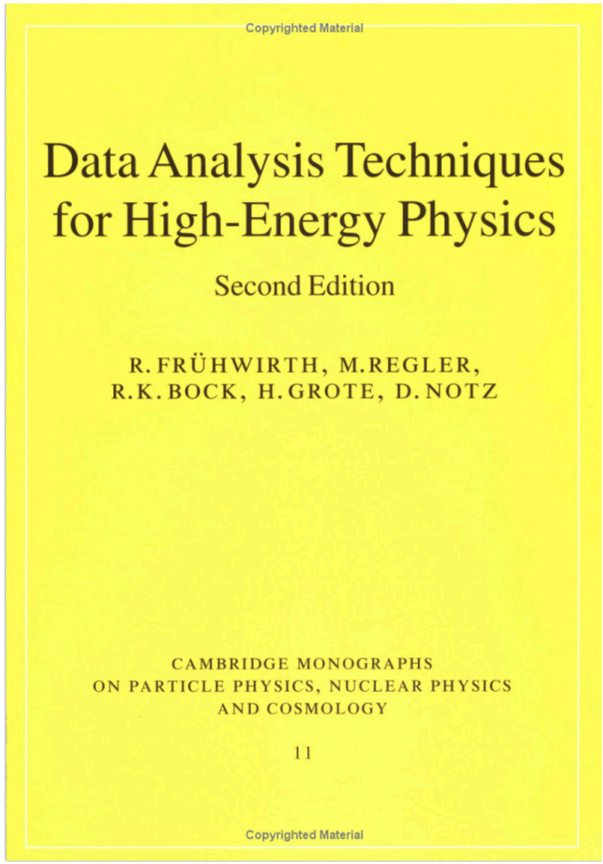
# Some global method ideas

Described here:

- **Hopfield network**
- **Clustering**
- **Sorting methods**

More in literature:

- Template matching
- Hough transform / Histogramming method
- Minimum Spanning Tree

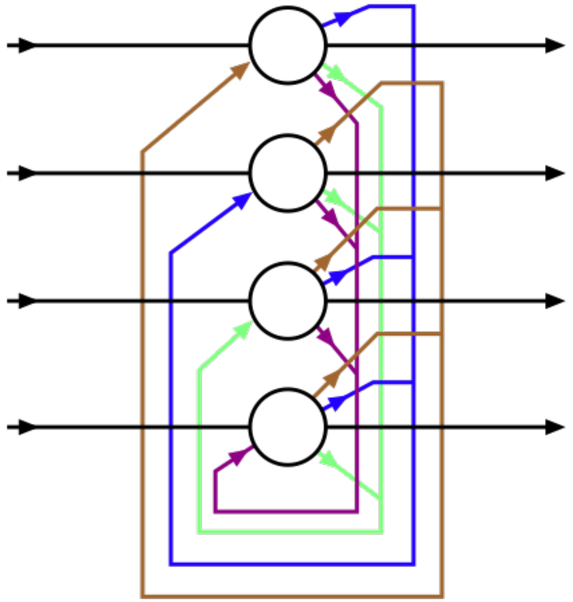


# Global methods for track reconstruction

- Hopfield network
- Clustering
- Sorting methods

# Hopfield network

- It is a recursive artificial neural network (RNN).
- It consists of binary neurons, and is fully connected.
- Connection weights $T_{ij}$ are symmetric, diagonal is zero.
- Updating is typically asynchronous, and follows:



$$s_i' = \Theta(\sum_{j=1}^{n} T_{ij} s_j - t_j)$$

with states $s$, a threshold $t_j$ and with $\Theta$ being the *Heaviside function:*

$$\Theta(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

# Hopfield network (2)

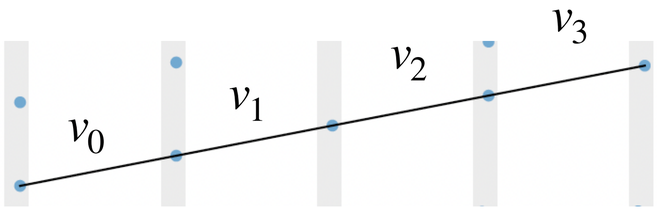
- Asynchronous updating leads to a stable state:

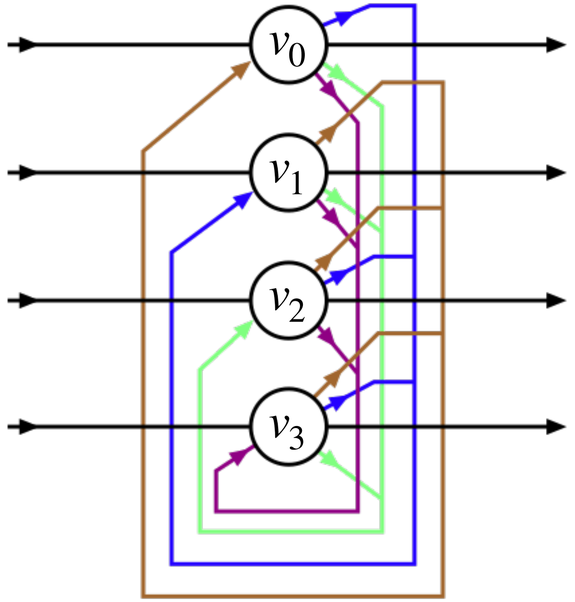
$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij}s_i s_j \qquad \Delta E \leq 0$$

- Hopfield networks *can solve general combinatorial optimization problems, provided that the objective function is quadratic with zero diagonal*.

# Hopfield network for track reconstruction

- Binary neurons represent short track segments connecting hits.
- Weights $T_{ij}$ should be chosen such that in the final stable state exactly those neurons are active whose corresponding track segments belong to a track.
- Generating all possible candidates leads to $O(n^2)$ neurons and $O(n^4)$ weights, where $n$ is the total number of hits.
- Some geometric cuts can be done according to detector geometry and angle of acceptance.



# Early results - Hopfield network

- Efficiency:

| Particle category | Reco eff. | Clone fraction | Hit purity | Hit eff. |
|---|---|---|---|---|
| All | 75.0 | 10.53 | 99.56 | 82.93 |
| Long | 92.5 | 17.58 | 99.51 | 79.77 |
| Long > 5 GeV | 95.0 | 18.41 | 99.45 | 79.97 |
| Long Strange | 85.8 | 10.06 | 99.48 | 88.52 |
| Long Strange > 5 GeV | 90.4 | 5.88 | 99.28 | 90.43 |
| Overall fake fraction | | 1.9 | | |

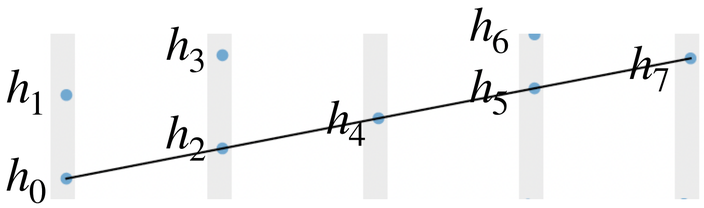
- Next steps:
  - Physics optimization (change update function, hyper param tuning).
  - Optimize algorithm (less segments, etc.)

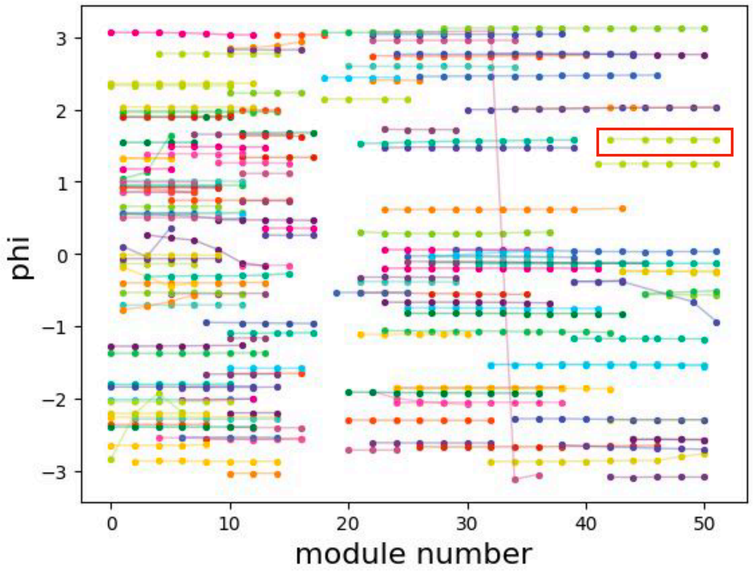
# Global methods for track reconstruction

- Hopfield network
- Clustering
- Sorting methods

# Clustering

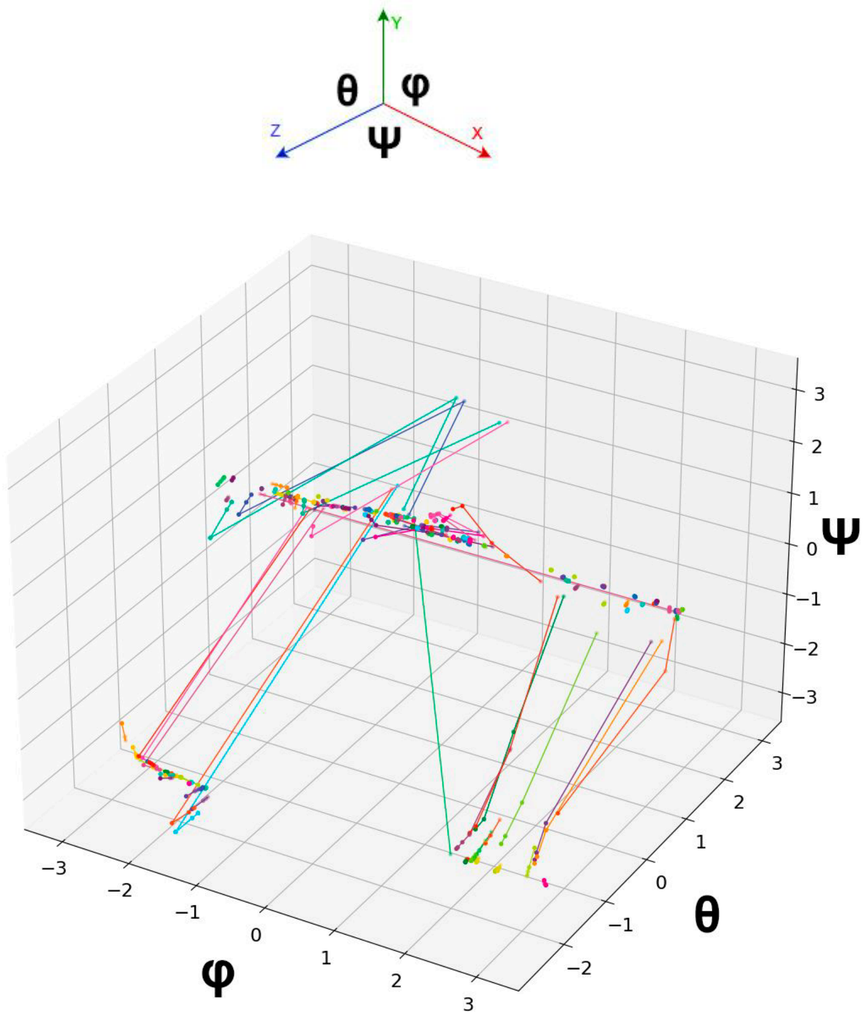
- Clustering consists in grouping together elements in space by some distance metric.
- For track reconstruction: cluster → hit.
- By choosing wisely the representation one prepares that *hits of tracks* end up close together.

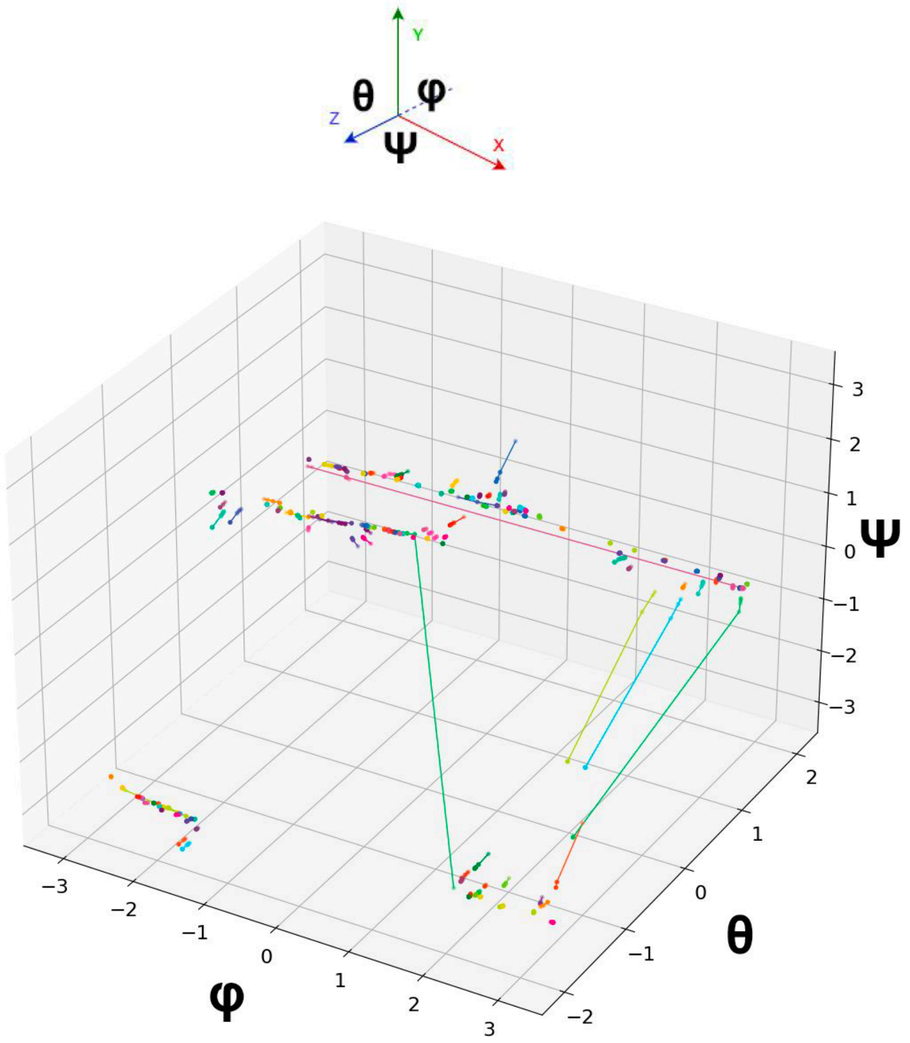



$h_0$ $h_2$ $h_4$ $h_5$ $h_7$

# Clustering - polar angles



# Adjusting distances

- Avoid hits in the same module by setting a high distance.
- Use the difference in hits' polar angle.
- Use Euclidean distance in 3D space.
- Test various methods for clustering (mean shift, affinity propagation, agglomerative clustering, HDBSCAN).

# Early results - clustering

- Efficiencies of different algorithms:

| Algorithm | Reco eff. | Clone fraction | Hit purity | Hit eff. | Fake fraction |
|---|---|---|---|---|---|
| Mean shift | 66.1 | 18.36 | 98.54 | 69.62 | 78.4 |
| HDBSCAN | 55.5 | 3.99 | 93.66 | 89.68 | 28.5 |
| Agglomerative clustering | 62.6 | 2.49 | 98.4 | 83.17 | 81.7 |

- Possible improvements:
  - Use segments instead of single hits.
  - Test with other projection spaces, distance metrics.

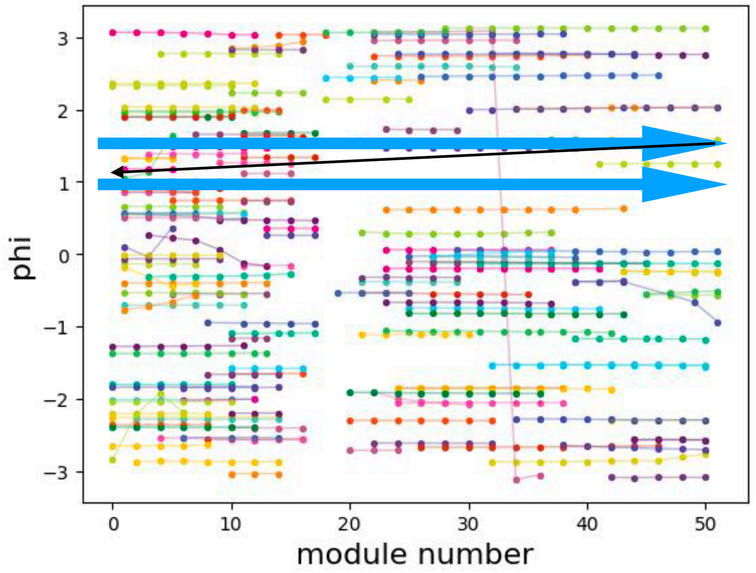
# Global methods for track reconstruction

- Hopfield network
- Clustering
- Sorting methods

# Sorting method

- *Partial solutions* are also interesting: solve a big chunk of the problem first, then look at rest of data.

- **If data is sorted** such that hits in tracks lie close together in memory, then tracking would simply consist in traversing a list in $O(n)$



# Early results - sorting

- Efficiency:

| Particle category | Reco eff. | Clone fraction | Hit purity | Hit eff. |
|---|---|---|---|---|
| All | 76.1 | 6.40 | 99.68 | 79.45 |
| Long | 85.8 | 10.34 | 99.72 | 74.64 |
| Long > 5 GeV | 86.8 | 7.93 | 99.76 | 76.43 |
| Long Strange | 38.7 | 11.77 | 99.77 | 63.31 |
| Long Strange > 5 GeV | 43.6 | 11.45 | 99.82 | 65.44 |
| Overall fake fraction | | 1.7 | | |

- This method requires a separate traversal / tracking step $O(n)$.
  - The traversal can keep various "forming tracks" and check them on the go.
- It has potential to act as a first stage of reconstruction.

# Conclusions

- Velo track reconstruction remains one of the **most relevant** problems in LHCb reconstruction.
  - With every new iteration of the detector this will remain the case.
- Many methods have been explored during the years, both *local* and *global*.
- Several options to pursue, possible gains in **efficiency** and **scalability**.
- The door for new architectures is open!

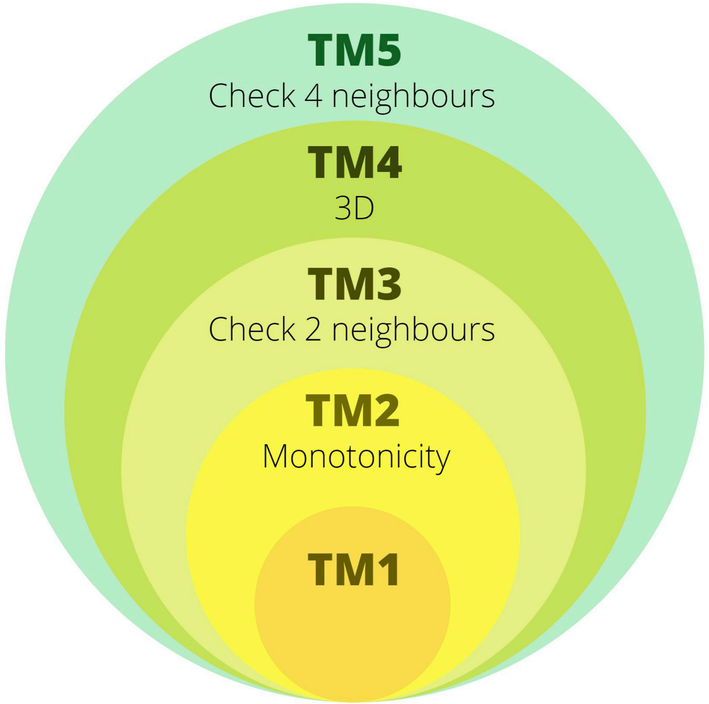
# Backup

# Global methods for track reconstruction

- Hopfield network
- Clustering
- Sorting methods
- Template matching

# Template matching

- A dictionary of all possible "classes" of tracks is required.




**Modules**

| **Templates** | Module 1 | Module 2 | Module 3 | ... |
|---|---|---|---|---|
| Temp. 1 | 0 | hit 1 | 0 | |
| Temp. 2 | 0 | 0 | 0 | |
| Temp. 3 | 0 | 0 | hit 6 | |
| | | | | |
| ⋮ | | | | |

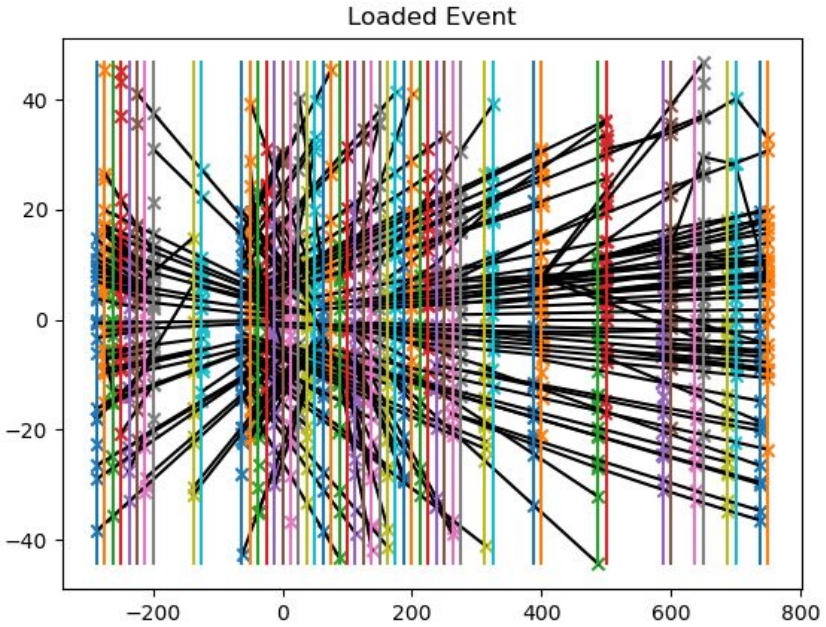
# Template matching - early results

- Efficiencies:

| Templates | Reco. eff | Clone fraction | Hit purity | Hit eff. | Ghost rate |
|---|---|---|---|---|---|
| 700 | 64.6 | 17.2 | 97.1 | 67.5 | 13.3 |
| 800 | 63.5 | 18.3 | 97.2 | 65.8 | 13.1 |
| 900 | 62.7 | 19.3 | 97.4 | 64.2 | 12.5 |
| 1000 | 61.8 | 19.9 | 97.6 | 62.9 | 12.1 |

- Similarly to clustering, this could be a first filter.

- It is typically used in *low redundancy environments* (with a few detector elements), which the Velo does not fit into.




Minibias Dataset: Event 113

1 Load event

2 Separate odd/even

3 Initialise neurons

4 Initialise weights

5 Converge to local minimum

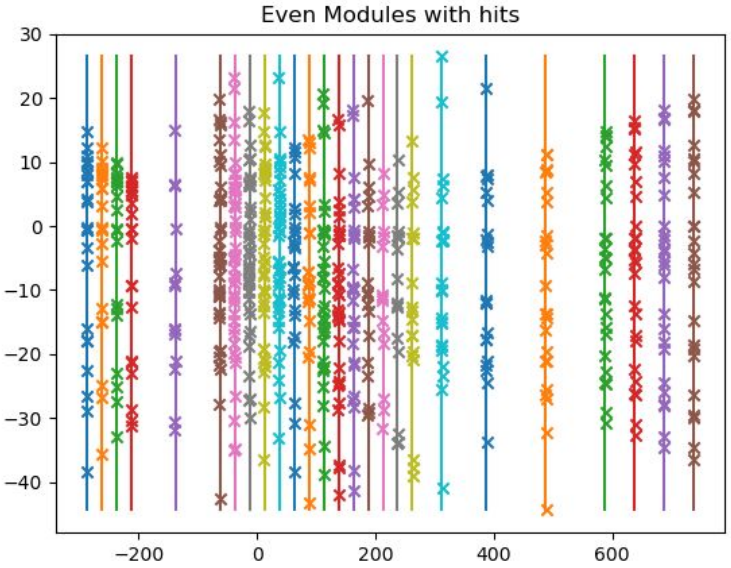
6 Repeat 10 times

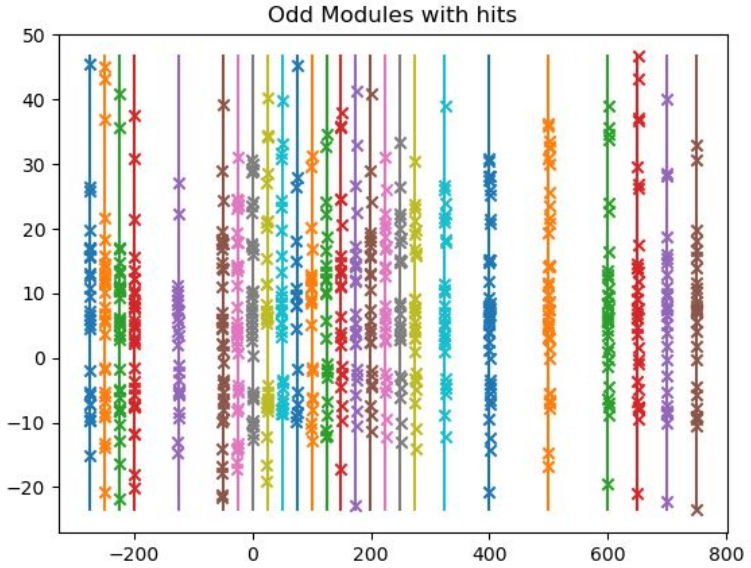
7 Average

8 Bifurcation removal

9 Prune tracks

10 Combine to final output



1 Load event

2 Separate odd/even

3 Initialise neurons

**activation range: [0,1]**

4 Initialise weights

5 Converge to local minimum

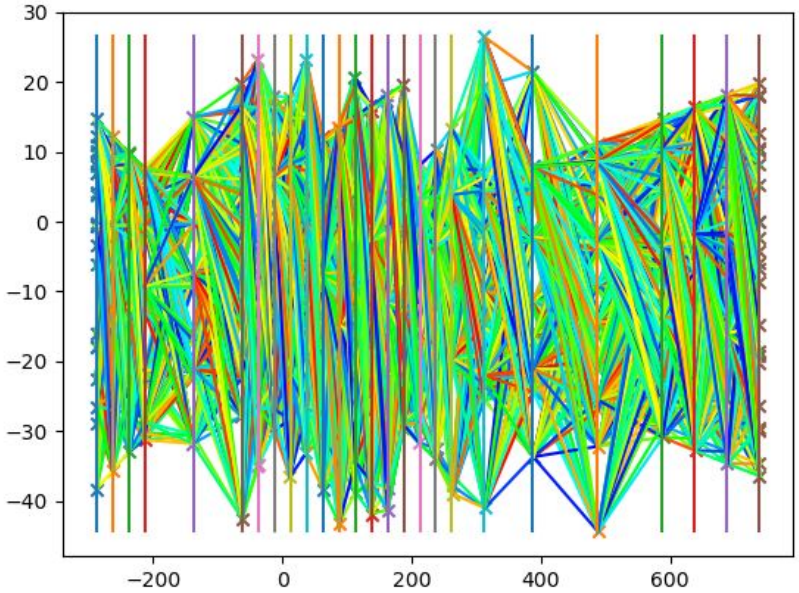
6 Repeat 10 times

7 Average

8 Bifurcation removal

9 Prune tracks

10 Combine to final output



1 Load event

2 Separate odd/even

3 Initialise neurons

**activation threshold: 0.2**

4 Initialise weights

5 Converge to local minimum

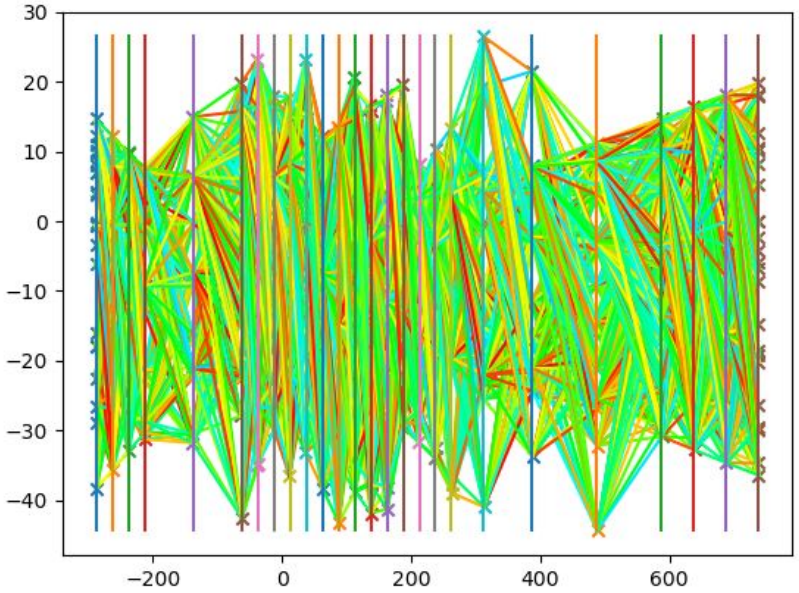
6 Repeat 10 times

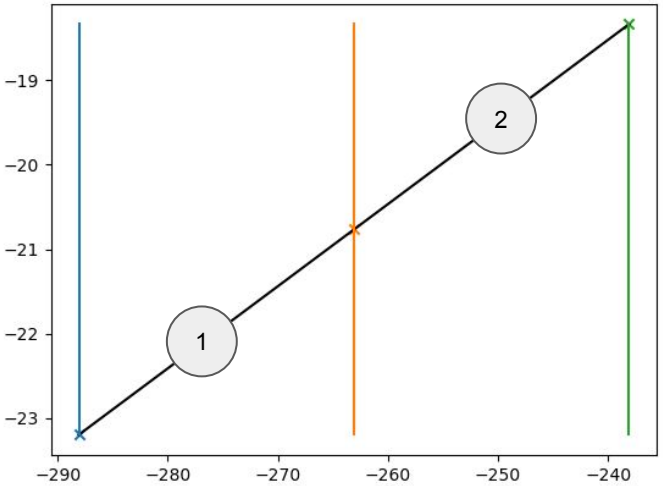
7 Average

8 Bifurcation removal

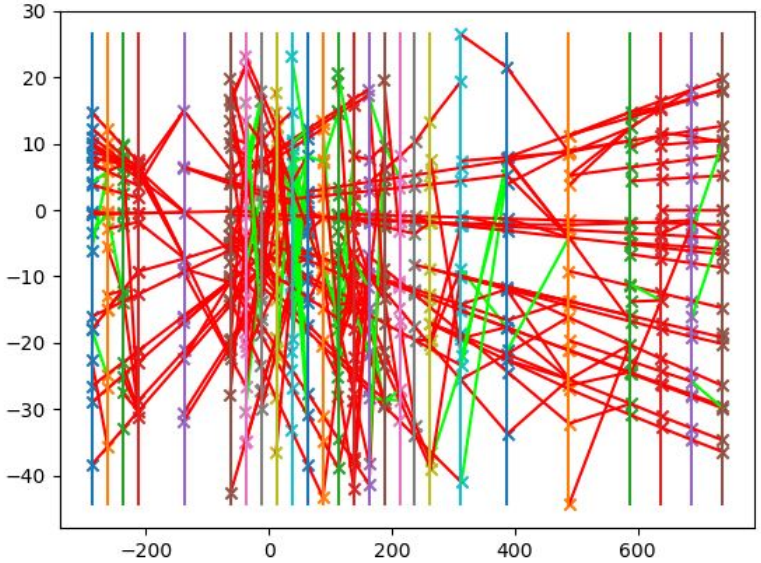
9 Prune tracks

10 Combine to final output






$$w_{12} = \max(0, \alpha(1 - \sin\theta_{12})^{\beta}(1 - \sin\varphi_{12})^{\gamma} + \Lambda_{12} + \Lambda_{12}^{*})$$




Energy: -576.92

1 Load event

2 Separate odd/even

3 Initialise neurons

4 Initialise weights

5 Converge to local minimum

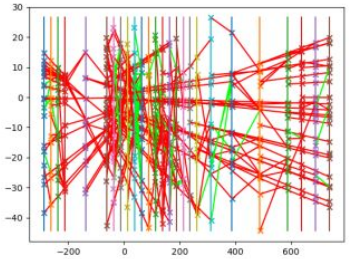
6 Repeat 10 times

7 Average

8 Bifurcation removal

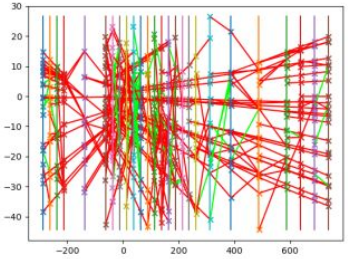
9 Prune tracks

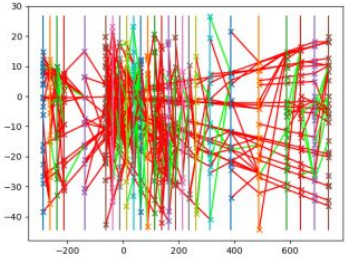
10 Combine to final output

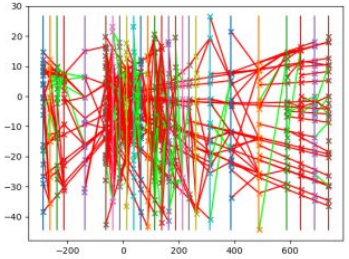


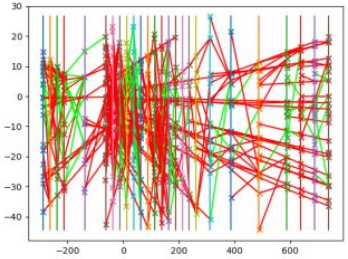

Energy: -576.92

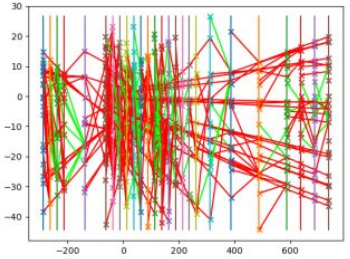
1 Load event | 2 Separate odd/even | 3 Initialise neurons | 4 Initialise weights | 5 Converge to local minimum | **6 Repeat 10 times** | 7 Average | 8 Bifurcation removal | 9 Prune tracks | 10 Combine to final output

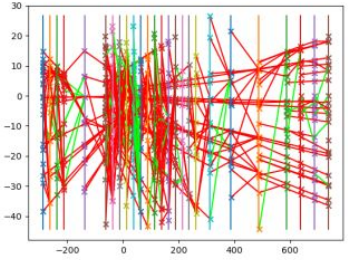

Energy: -576.92

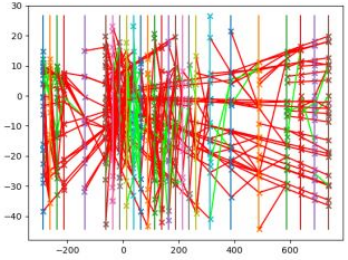

Energy: -357.10

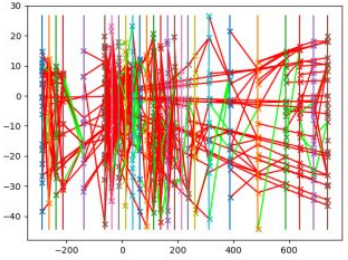

Energy: -401.79

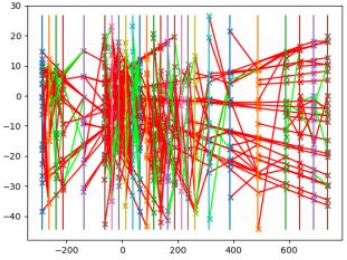

Energy: -426.68

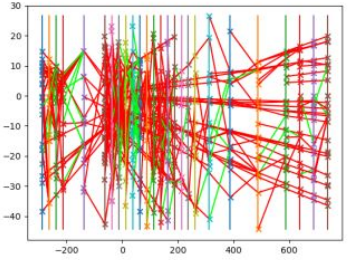

Energy: -418.97


Energy: -406.88


Energy: -453.28


Energy: -371.40


Energy: -420.67


Energy: -443.03

1 Load event

2 Separate odd/even

3 Initialise neurons

4 Initialise weights

5 Converge to local minimum

6 Repeat 10 times

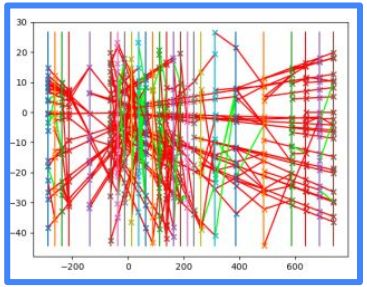
7 Average

- **below_mean**
- below_median
- minimum
- complete

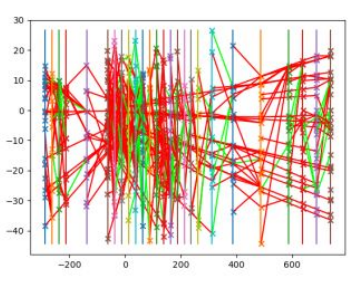
8 Bifurcation removal

9 Prune tracks

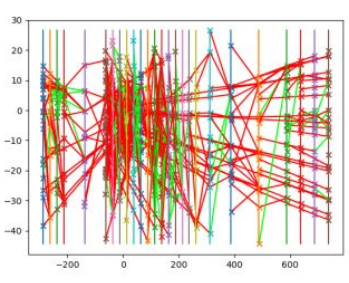
10 Combine to final output



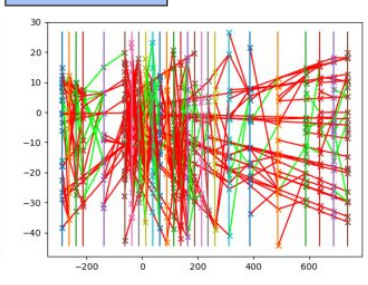
**Energy: -576.92**



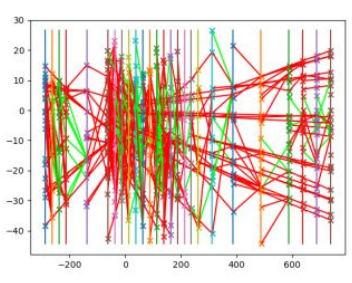
Energy: -357.10



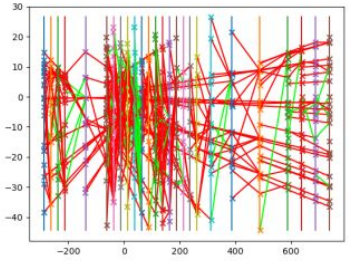
Energy: -401.79



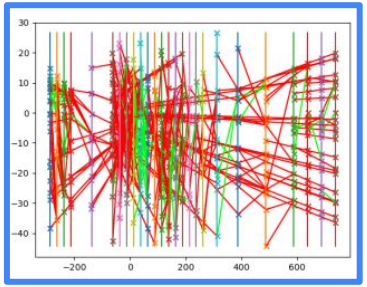
Energy: -426.68



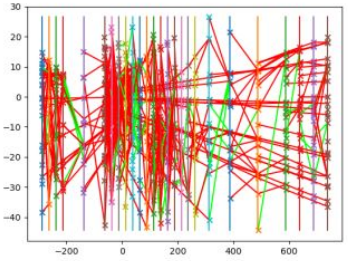
Energy: -418.97



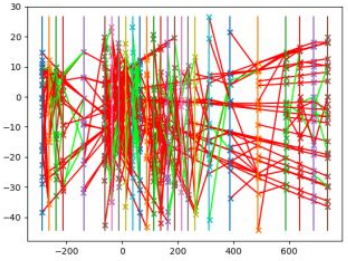
Energy: -406.88



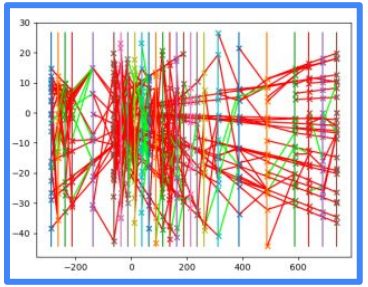
**Energy: -453.28**



Energy: -371.40



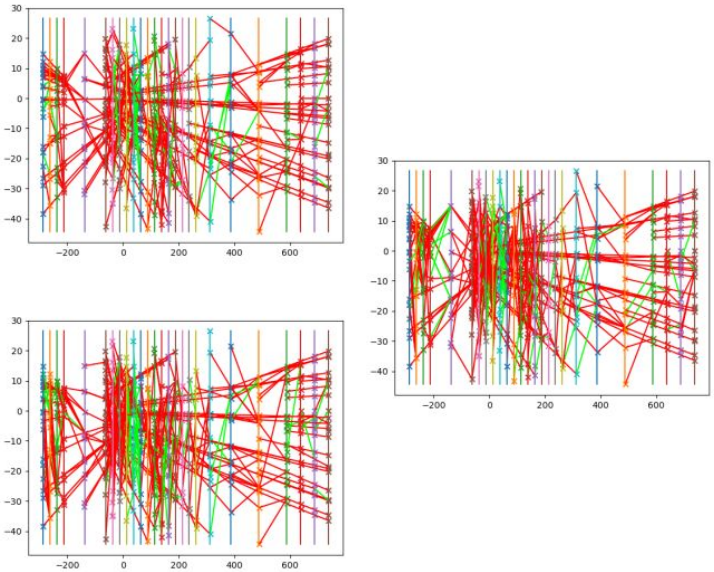
Energy: -420.67

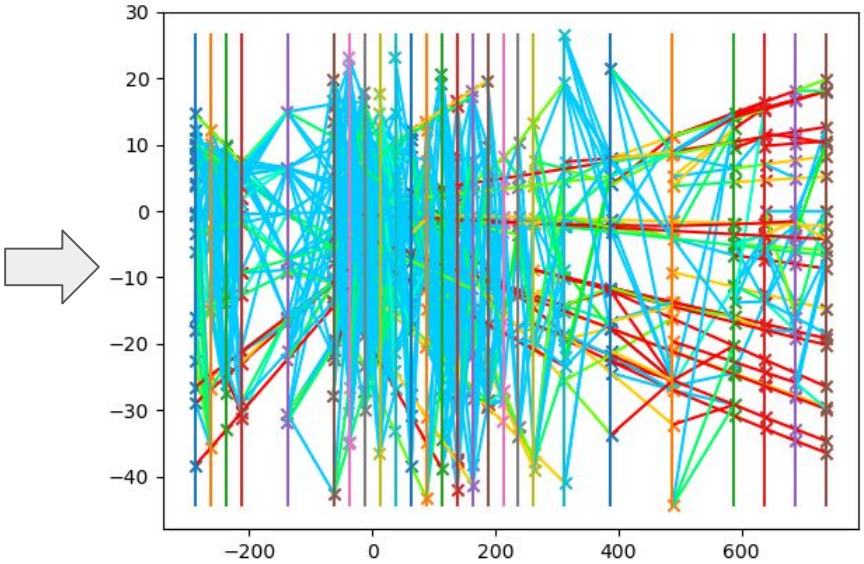


**Energy: -443.03**

Mean Energy: -427.67

Maastricht University





1 Load event

2 Separate odd/even

3 Initialise neurons

4 Initialise weights

5 Converge to local minimum

6 Repeat 10 times

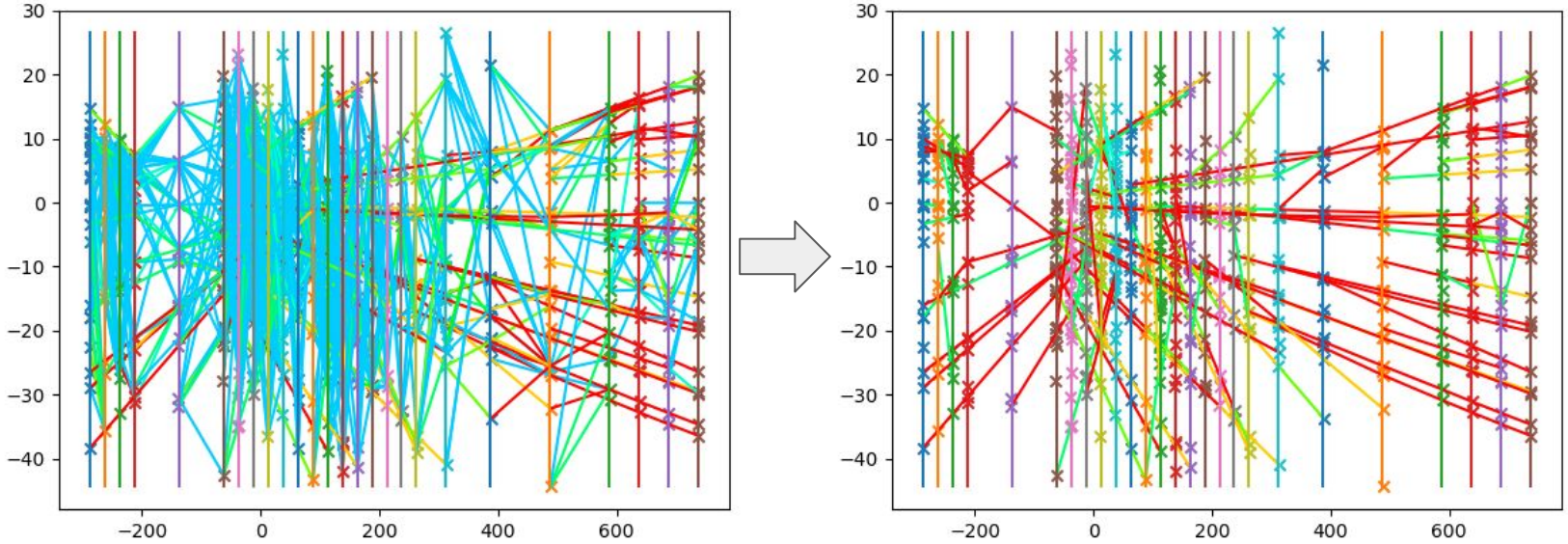
7 Average

8 Bifurcation removal

- max_activation
- **smart removal**

9 Prune tracks

10 Combine to final output



1 Load event

2 Separate odd/even

3 Initialise neurons

4 Initialise weights

5 Converge to local minimum

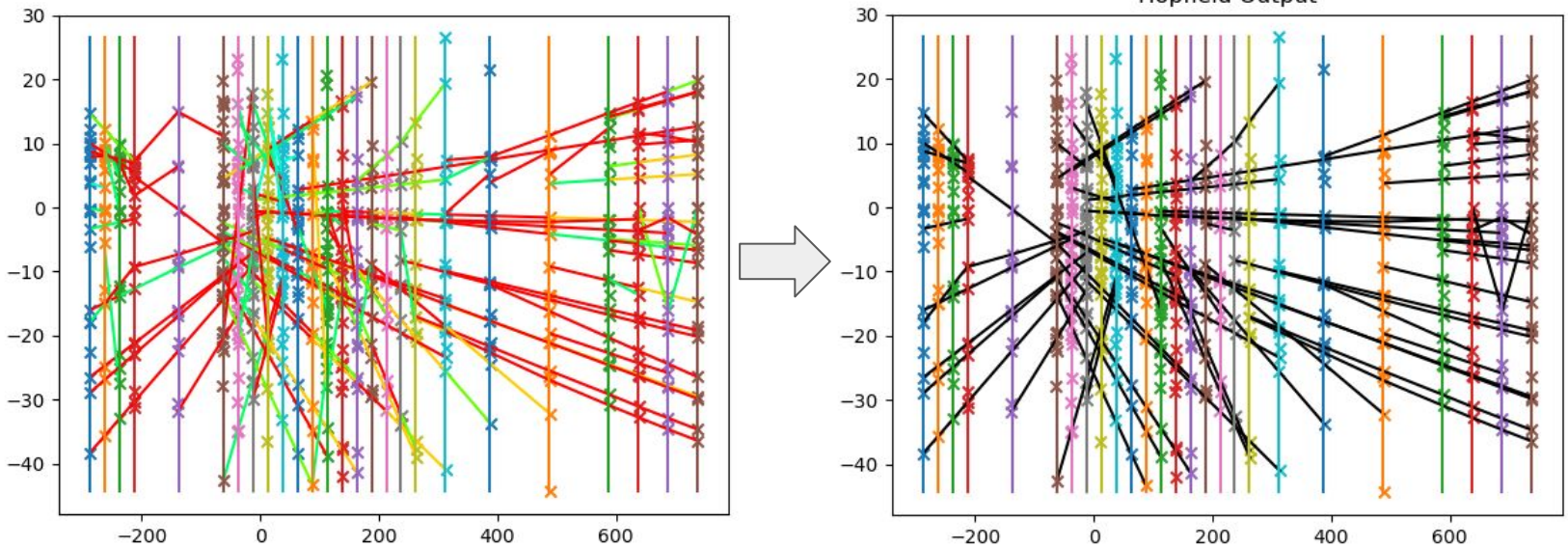
6 Repeat 10 times

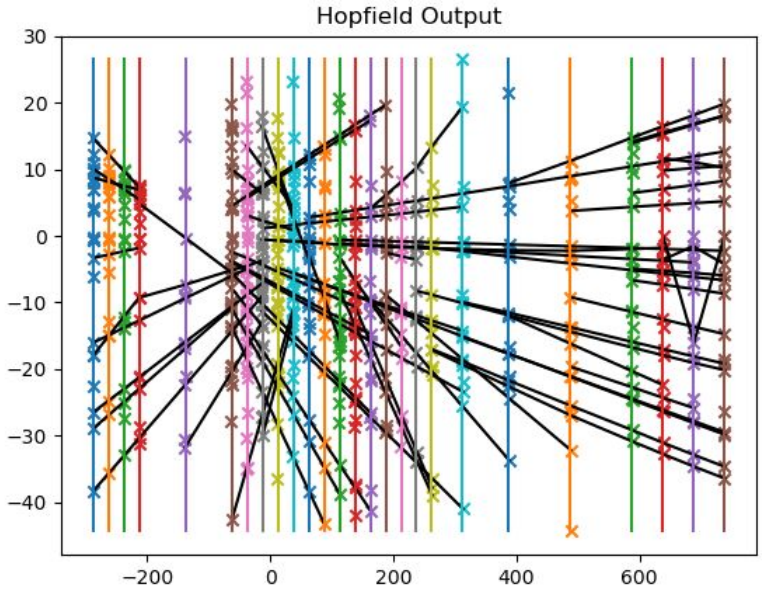
7 Average

8 Bifurcation removal

9 Prune tracks

10 Combine to final output






Even Hopfield Output

1 Load event

2 Separate odd/even

3 Initialise neurons

4 Initialise weights

5 Converge to local minimum

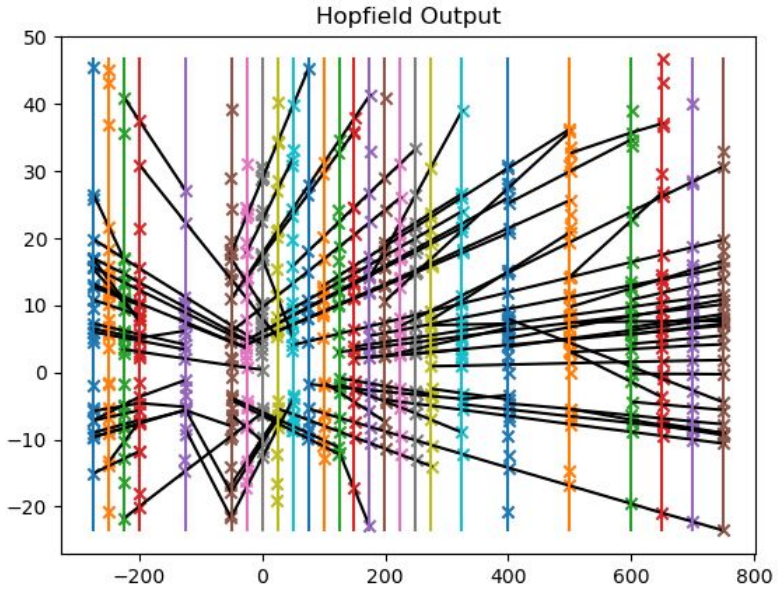
6 Repeat 10 times

7 Average

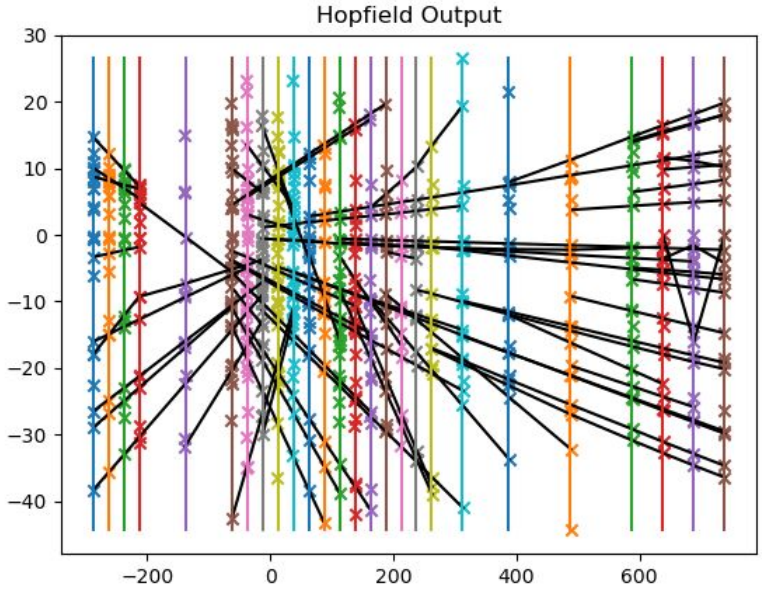
8 Bifurcation removal

9 Prune tracks

10 Combine to final output




Odd Hopfield Output




Even Hopfield Output

1 Load event

2 Separate odd/even

3 Initialise neurons

4 Initialise weights

5 Converge to local minimum

6 Repeat 10 times

7 Average

8 Bifurcation removal

9 Prune tracks

10 Combine to final output