Self Organizing Systems 2022W
Assignment 3:

Group Coding Topic A Dendrogramm:

Blohm Peter	11905150
Braunsperger Martin	11909911
Gawor Adrian Jan	11905152

https://github.com/Pietreus/sos-2022w-exercise3

Abstract

The goal of this assignment was to implement a dendrogram visualization of the hierarchical clustering of SOM weights. Besides a dendrogram visualization we implemented a minimal spanning tree visualization too. Furthermore, we trained datasets using the Java SOM Toolbox and MiniSOM which we then visualized using our implementation and the Java SOM Toolbox's implementation, at least where there was a direct correspondence. We also visualize the trained data of the Java SOM Toolbox and MiniSOM using our implementation and compare slight differences.

For parameter testing purposes we generated an exhaustive set of visualizations, which can be found in the clustering_vis_plots folder in our repository.

1 Implementation

1.1 Training

The function

```
def train_som(path,
              x,
              y,
              iterations):
```

is used for loading and training the SOM weights. After loading the weights we pass them into MiniSOM and align the SOM according to the PCA axes and train it using the given amount of iterations. We then transform the resulting trained SOM into the same format used by the Java SOM Toolbox. We noticed that the format used by the Java SOM Toolbox is either weirdly designed or erronous. For example the 'xdim' property actually corresponds to 'y' and vice versa. Furthermore the Java SOM Toolbox does not use a 3D array for the weights but an 'unwrapped' 2D array representing the slices of the 3D structure.

To ensure that our training procedure using MiniSOM is correct, we compare its result to the models trained using the SOM Toolbox. Both models are visualized using our implementation.

We can see that the models certainly are not identical, but the clustering produces similar clusters which topographically are roughly in the same places. Besides slight differences in the cluster positions, we can observe smoother cluster boundaries in the data trained using the Java SOM Toolbox.

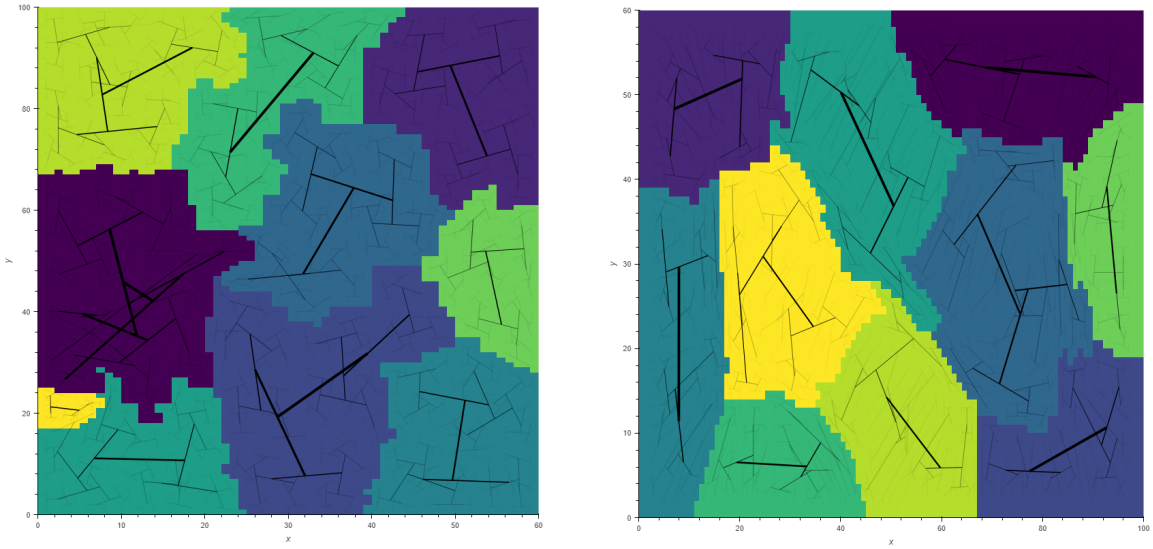
1.2 Visualization

The function

```
def clustering_vis(weights,
                   n_clust,
                   linkage,
                   line_distance_cutoff,
                   max_width,
                   inter_cluster = True,
                   intra_cluster = False,
                   line_weighting_method = "size"):
```

is used as the actual main function used for setting the visualization parameters, generating the lines of the respective visualization type and drawing them on top of the clustered image (in the case of the dendrogram at least).

We use the 'Clustering' function and some helper functions described in the next sub-sections, to generate the information needed for drawing a dendrogram or MST.



Self-trained model (MiniSOM) SOM Toolbox model

Figure 1: Comparison of SOM models for the chainlink dataset based on our visualization

1.3 Clustering

The function

```
def Clustering(m,
               n,
               weights,
               n_clusters=2,
               linkage='ward',
               line_distance_cutoff = 10,
               max_width = 0.8,
               inter_cluster = False,
               intra_cluster = True,
               line_weighting_method = None):
```

is used to first agglomeratively cluster the trained SOM map weights, to generate the lines of either the dendrogram or the minimal spanning tree. The most interesting parameters of this function are:

1. linkage, which determines the linkage type used for clustering. It may be one of:
   a) 'single', which uses single linkage
   b) 'average', which uses average linkage
   c) 'complete', which uses complete linkage
   d) 'ward', which uses ward linkage
   e) 'mst', if set does not perform clustering and rather just generates the lines of a minimal spanning tree
2. line_distance_cutoff, which determines the maximum distance of a line in the dendrogram before it get's omitted.
3. max_width, which acts as a factor for the line width of the MST and dendrogram
4. inter_cluster, which when true enables drawing of branches between clusters, i.e. the dendrogram structure from the root node down to the cluster centers.
5. intra_cluster, which when true enables drawing of branches inside individual clusters, i.e. the dendrogram structure from the cluster centers down to it's leaf nodes.
6. line_weighting_method, which may be one of:
   a) None, which sets the line width for every line to the maximum possible width
   b) 'size', which sets the line width in relation to the cluster size
   c) 'distance', which sets the line width in relation to the length/distance of the line, where a large distance yields a thicker line

d) 'inverse_distance', which acts in a similar fashion to 'distance' just that a small distance yields a thinner line

1.4 Dendrogram

The function

```
def get_dendrogram_lines(m,
                         n,
                         children,
                         distances,
                         num_samples,
                         n_clusters,
                         inter_cluster_lines,
                         intra_cluster_lines,
                         line_weighting_method,
                         line_distance_cutoff,
                         max_width):
```

is used to get information about the lines representing the dendrogram (i.e. starting position, ending position and width). The most interesting parameters of this function are:

1. linkage, which determines the linkage type used for clustering. It may be one of:
   a) 'single', which uses single linkage
   b) 'average', which uses average linkage
   c) 'complete', which uses complete linkage
   d) 'ward', which uses ward linkage
2. line_distance_cutoff, which determines the maximum distance of a line in the dendrogram before it get's omitted.
3. max_width, which acts as a factor for the line width of the dendrogram
4. inter_cluster, which when true enables drawing of branches between clusters, i.e. the dendrogram structure from the root node down to the cluster centers.
5. intra_cluster, which when true enables drawing of branches inside individual clusters, i.e. the dendrogram structure from the cluster centers down to it's leaf nodes.
6. line_weighting_method, which may be one of:
   a) None, which sets the line width for every line to the maximum possible width
   b) 'size', which sets the line width in relation to the cluster size
   c) 'distance', which sets the line width in relation to the length/distance of the line, where a large distance yields a thicker line
   d) 'inverse_distance', which acts in a similar fashion to 'distance' just that a small distance yields a thinner line

This function takes in the clustering information, which contains the "branchings" i.e. the nodes where one cluster splits up into two subclusters. This essentially gives us the information that we need to draw the respective dendrogram lines. We iterate through all these "branchings" giving us child1, child2 and their positions. These can then be used for calculating the distance between the two subclusters, which in turn can be transformed into the line width used for connecting the two subclusters. Finally, based on the positions and calculated width we can fill the lines array with the current line for later drawing.

1.5 Minimal Spanning Tree

The function

```
def get_mst_lines(m,
                  n,
                  weights,
                  method,
                  max_width):
```

is used to get information about the lines representing the minimal spanning tree (i.e. starting position, ending position and width). The most interesting parameters of this function are:

1. method, which may be one of:
    a) None, which sets the line width for every line to the maximum possible width
    b) 'size', which sets the line width in relation to the cluster size
    c) 'distance', which sets the line width in relation to the length/distance of the line, where a large distance yields a thicker line
    d) 'inverse_distance', which acts in a similar fashion to 'distance' just that a small distance yields a thinner line
2. max_width, which acts as a factor for the line width of the MST

For computing a MST we use the package 'networkx', which contains an own implementation of a graph datastructure. Therefore, we first create a fully-connected with the edge weight between two vertices being the euclidean distance between their positions in the trained SOM. Using the function 'networkx.algorithms.tree.minimum_spanning_edges' we can compute this fully-connected graph. As this graph is potentially very large in size due to the amount of vertices and edges, this step may **take several minutes depending on the size of the SOM**.
This cannot be really avoided as MST construction in high-dimensional spaces can be done only marginally faster than $O(n^2)$.

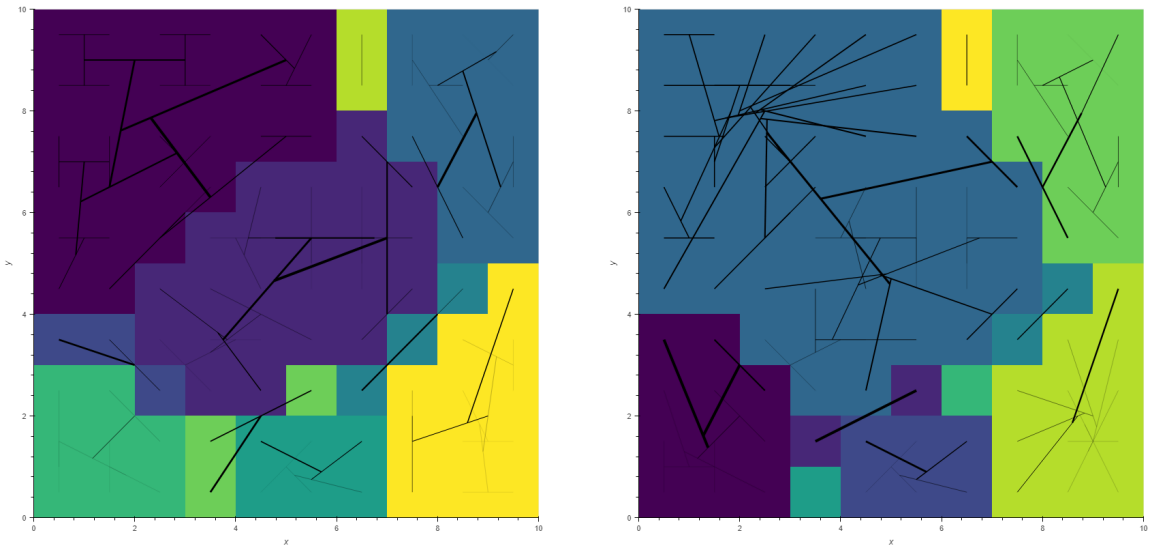
After successful computation of a MST, we iterate through all contained edges and using their positions and euclidean distances/lengths we can add lines with a given length based on the selected method.

2 Visualizations

2.1 Line weighting methods

Since the thickness of the lines can be used to emphasize different aspects, we chose to include unweighted line thickness, as well as line thickness based on the cluster sizes, which both can act as defaults for clean visualisations depending on the size of the SOM.

In general, one would like neighboring clusters to be agglomerated, so there should be some inherent order visible in the trees and no excessive crossing of lines. Excessive crossing of lines hints to a unfitting clustering approach or topology violations, a good and bad example of the tree structure can be seen by comparing wards method to single linkage in figure 2



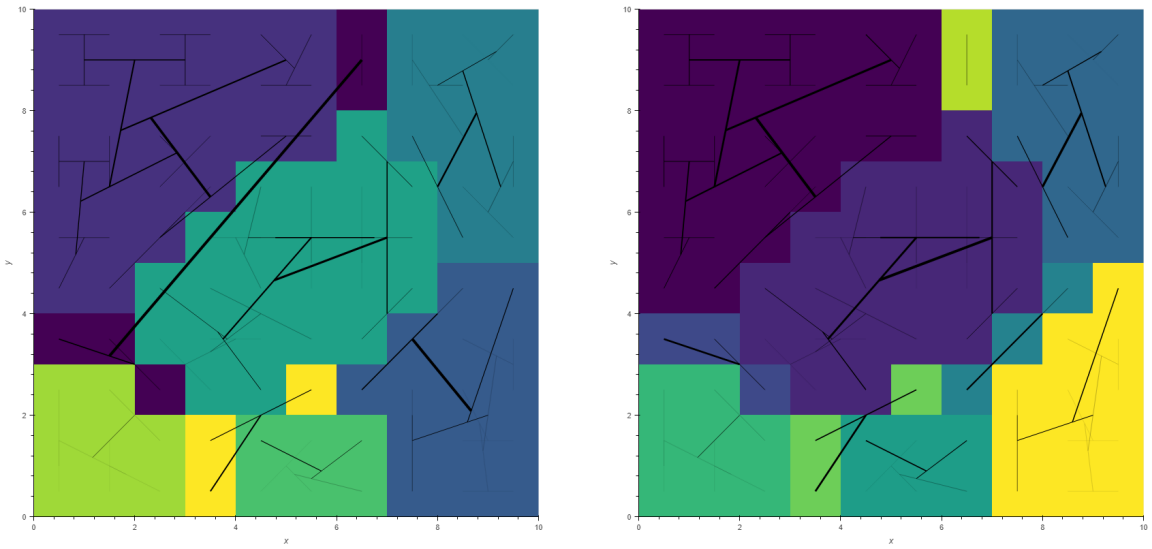
Ward Clustering Dendrogram | Single Linkage Clustering Dendrogram

Figure 2: Intra-cluster line overlays in Ward and single linkage for the 10 cluster dataset with 10 clusters. Clustering Dendrograms reveal much more stable clusters with ward method as opposed to single linkage

Not only that but the topology in different regions can be reviewed quite separately, as maybe some clusters are a great fit using the SOM, and some are not. Furthermore, the different structures within clusters might enable the differentiation of clusters from their dendrograms alone as can be seen in figure 2 as well, by looking at the different clusters in ward clustering. The largest cluster on the top left can

be seen to be much less dense than the one on the bottom left for example. Still, the dendrogram seems quite well structured.

Figure 3 shows clusters with high variability, which could indicate where clusters should be split. Also, a topology violation is revealed with the 8 cluster method, which is important information.

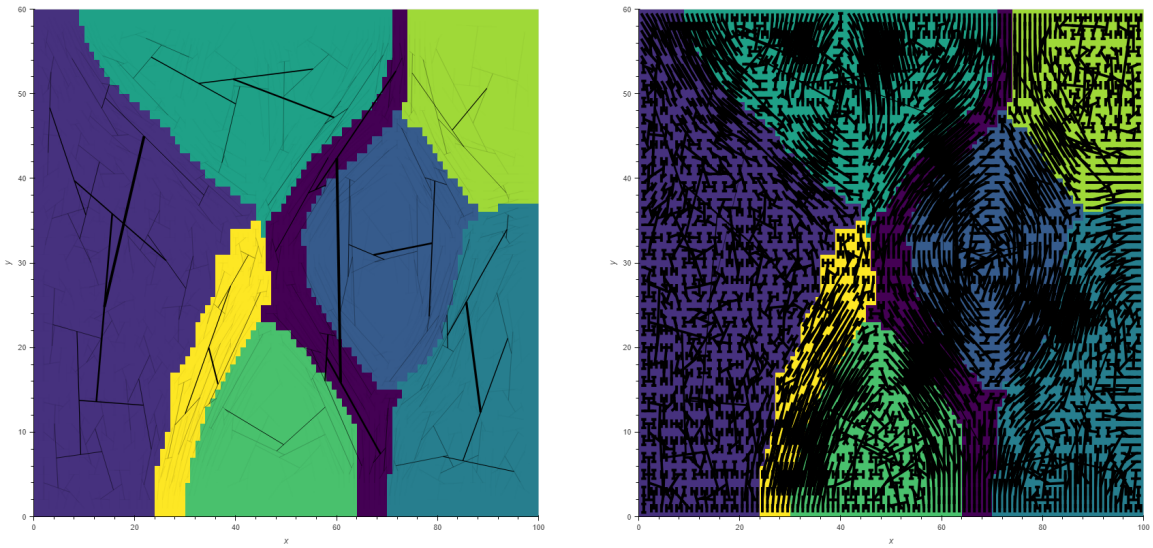


8 clusters 10 clusters

Figure 3: Intra-cluster line overlays in the ward method for the 10 cluster dataset with 8 and 10 clusters. Clustering Dendrograms reveal fault lines for the two clusters which should be separated

For interpretability of structure, distance weighting emphasizes large distances, so for many units this plot looks still manageable and the thick lines within clusters might indicate possible splitting points and thin lines between might indicate where clusters might be merged.

Inverse distances can effectively show how dense different clusters are, in the 10clusters dataset the different densities become very apparent.



Distance Inverse Distance

Figure 4: Line weighting using inverse distance weighting reveals the density of the clusters.

2.2 Inter- and Intracluster Dendrograms

Showing Cluster-dendrograms can be toggled separately for between and within the clusters, as the interpretation is different and showing both at once might be confusing or overwhelming for users.

Both show the cohesiveness of the clusters and the difference between them, depending on the line weighting method.

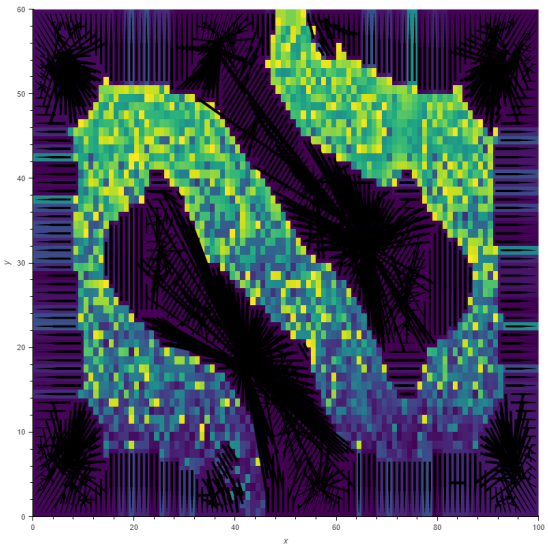
Intracluster Dendrograms can hint on the structure within the clusters, as described above, and can be used if two clusters are suspected to be split as in 3

2.3 Linkage Methods

The different linkage methods are the most commonly used agglomerative methods for clustering. Their advantages and disadvantages are well known and were already reviewed in the lectures and for many applications different methods are just compared to see which brings the most interpretable clustering.

Single linkage is a bit of a special case and works mostly for very few and very peculiar datasets. In this exercise, we noticed that the chainlink data set with many more clusters than initial data points in the large SOM shows the structure of the data very well, as the homogenous part of the SOM without any datapoints close to the units clusters first.

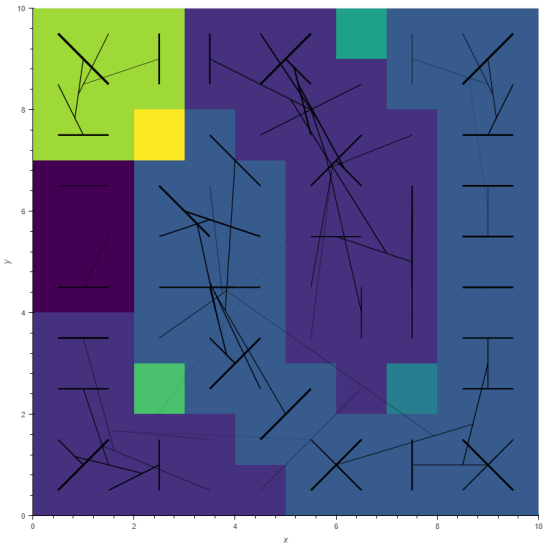
The result is a line overlay with interesting dendrogram overlays.



Intracluster dendrogram

Figure 5: Dendrogram of a single linkage clustering

In the small SOM, the clustering structure also is neatly visible with a cluster size of 8.

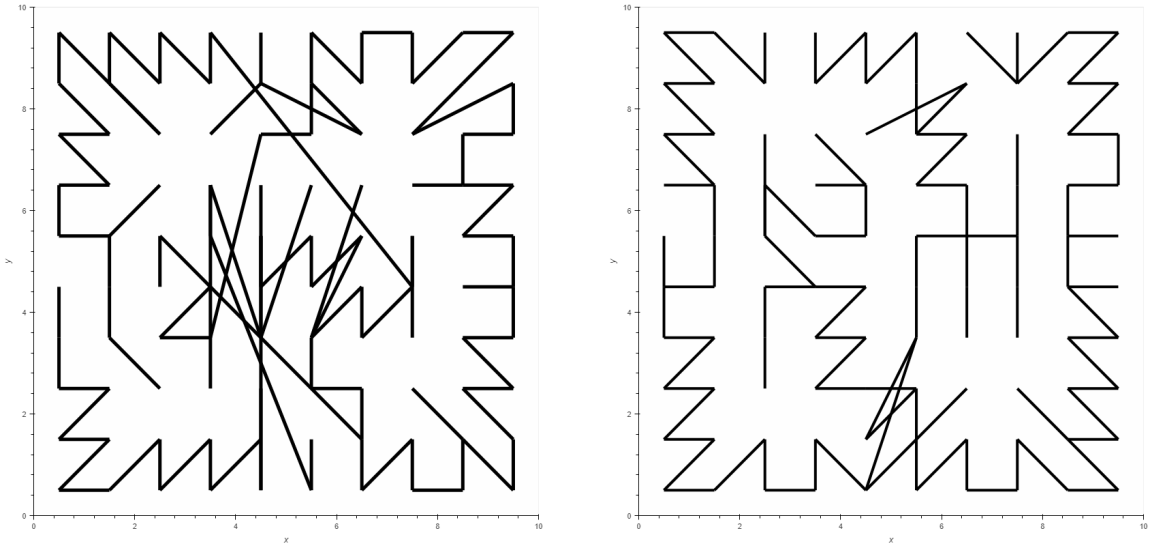


Intracluster dendrogram

Figure 6: Dendrogram of a single linkage clustering with a cluster size of 8

2.4 MST

MST and its interpretations were also covered in the lecture already. It can be used to find topology violations as in the self-trained small.



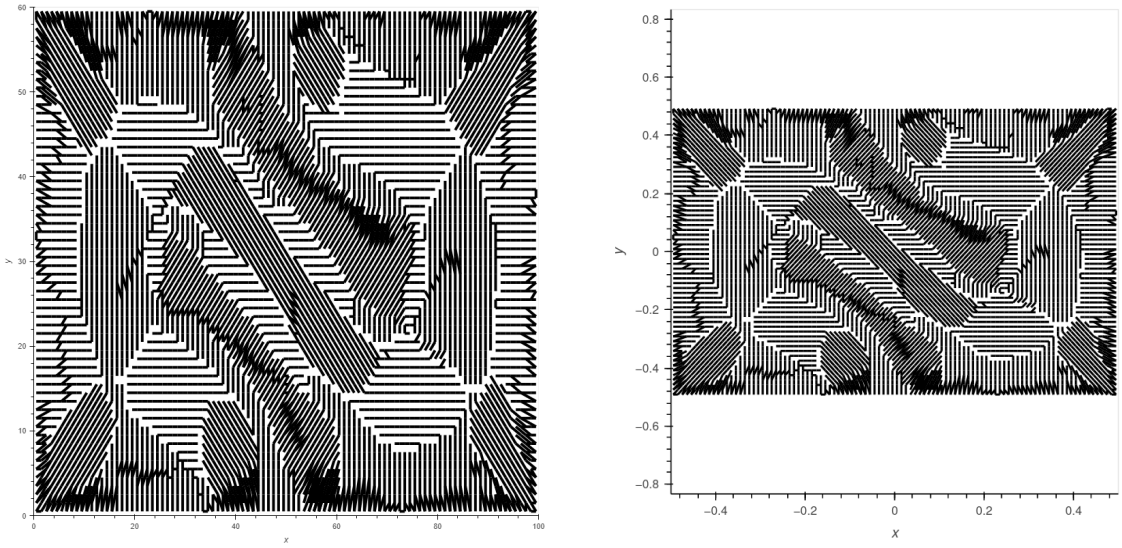
Our trained SOM | Java SOM Toolbox trained SOM

Figure 7: Comparison of differently trained MSTs

3 Comparison to SOM Toolbox

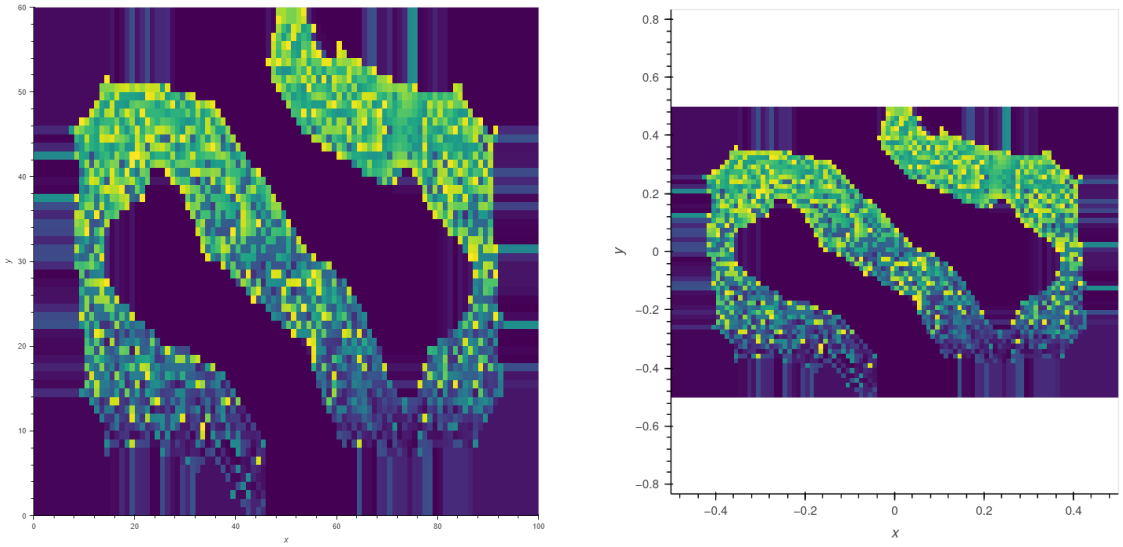
The Java SOM Toolbox does support neither a dendrogram visualizations nor clustering, as such we can only compare most of our results to those of PySOMVis. It does however include MST-based visualizations, which we can compare to both ours and those of PySOMVis. Unfortunately, the Java SOM Toolbox seems to be unable to handle the large SOMs, as it never finishes or crashes when trying to visualize them.

All the visualizations are basically identically in terms of clustering and MST, which shows that our implementation is correct (at least in those aspects).



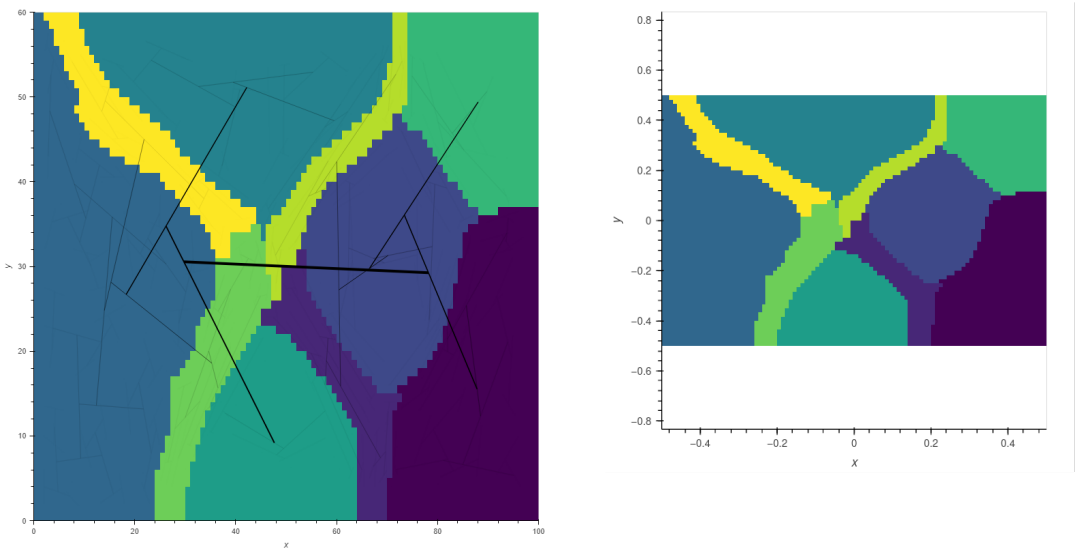
Our visualization | PySOMVis

Figure 8: Comparison of the visualizations using the large chainlink dataset based on its MST



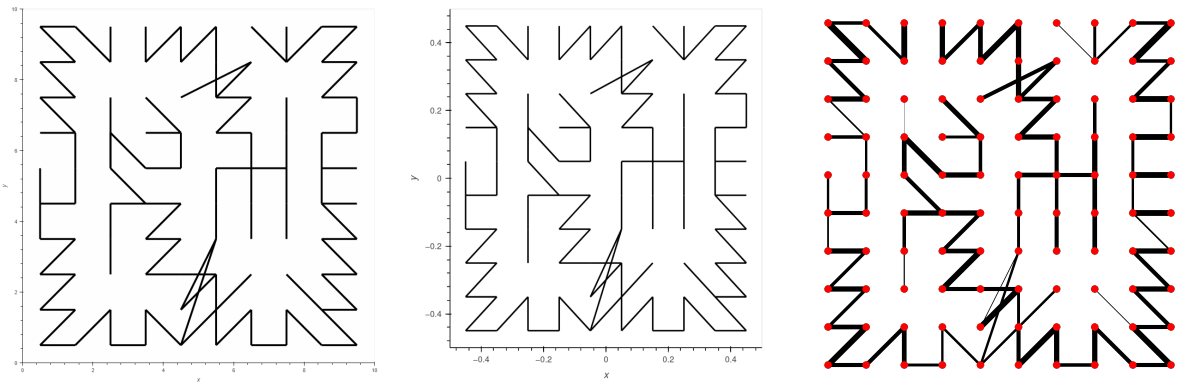
Our visualization PySOMVis

Figure 9: Comparison of the visualizations using the large chainlink dataset with single linkage and 2500 clusters



Our visualization PySOMVis

Figure 10: Comparison of the visualizations using the 10 clusters dataset with ward linkage and 10 clusters



Our visualization PySOMVis SOM Toolbox

Figure 11: Comparison of the visualizations using the small chainlink dataset based on its MST

4 Conclusion

Both the clustering and the MST are well explored methods for visualisation as well as for the specific field of SOMs.

The clustering line overlay essentially adds information about different possible choices and using different emphasis for the line weights and showing dendrograms everywhere, within or between clusters can be used to better understand the structure of the SOM and the data similar to the MST line overlay.