Tutorial on cell assembly detection at multiple time scales and lag constellations

Eleonora Russo, Daniel Durstewitz.

This tutorial provides a brief 'hands-on' introduction to the assembly detection methods proposed in *Russo & Durstewitz, 2016, Cell assemblies at multiple time scales with arbitrary lag constellations, eLife 5:e19428* (please cite this reference when using the present methods).

For the purpose of this tutorial, test sets of simulated spike trains were created in file `test_data.mat` as described in section *Construction of synthetical 'ground-truth' data* in Russo & Durstewitz, 2016 (in the following, references to specific paragraphs/sections and figures will always refer to this publication).

How to run the assembly detection routine:

1) Arrange spike trains from all recorded units in a matrix `spM`, where units run across rows and spike time stamps (not binned) across columns. Supernumerary matrix entries are to be filled with `NaN`.
   For this tutorial, load `test_data.mat` (Figure T1):

```
load('test_data.mat');
nneu=size(spM,1);                    % nneu := number of recorded units
```

2) Specify the vector of bin widths (Δ) to be tested:

```
BinSizes=[0.015 0.025 0.04 0.06 0.085 0.15 0.25 0.4 0.6 0.85 1.5];
```

   Note that `spM` and `BinSizes` have to be expressed in the same units of measurement (e.g. *sec*).

3) For each bin size specify the maximum lag l_{max} (in numbers of bins) to be tested (all lags within $[-l_{max}, l_{max}]$ will be tested):

```
MaxLags=[10  10  10  10  10  10  10  10  10  10  10];
```

Note that the method removes non-stationarity confounds on time scales $>\Delta l$, but still assumes the process to be reasonably stationary below that time scale. This implies that bin width Δ and l_{max} should not be chosen larger than really necessary to track processes of interest, since otherwise this may result in a violation of the method's assumptions.

4) Call `Main_assemblies_detection` function to run the detection algorithm (see section *Full function description* at the end of this manuscript for details on optional parameters and a description of the output structure `assembly`):

```
[assembly]=Main_assemblies_detection(spM,MaxLags,BinSizes);
```

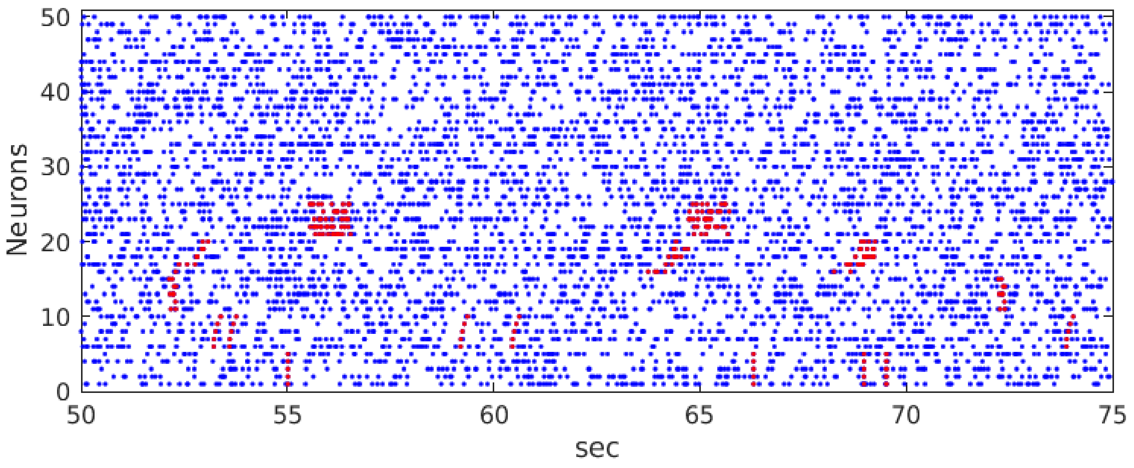

Figure T 1: Spike time rasterplot. Example of the spiking activity of 50 simulated units, of which 25 were assigned to 5 non-overlapping assemblies. Highlighted in red are the spikes belonging to the different assembly patterns.

Visualization

To visualize the detected assembly patterns, it is necessary to first create a new structure `As_across_bins`, in which assembly units are re-orderd according to their order of activation within the assembly:

```
[As_across_bins,As_across_bins_index]=assemblies_across_bins(assembly,BinSi
zes);
```

Using this structure, the assembly assignment matrix can now be displayed using function `assembly_assignment_matrix` (cf. Figs. 1, 3), which plots the assembly structures in terms of unit composition, lag constellations and temporal precision.

```
[Amatrix,Binvector,Unit_order,As_order]=assembly_assignment_matrix(As_acros
s_bins, nneu, BinSizes, display);
```

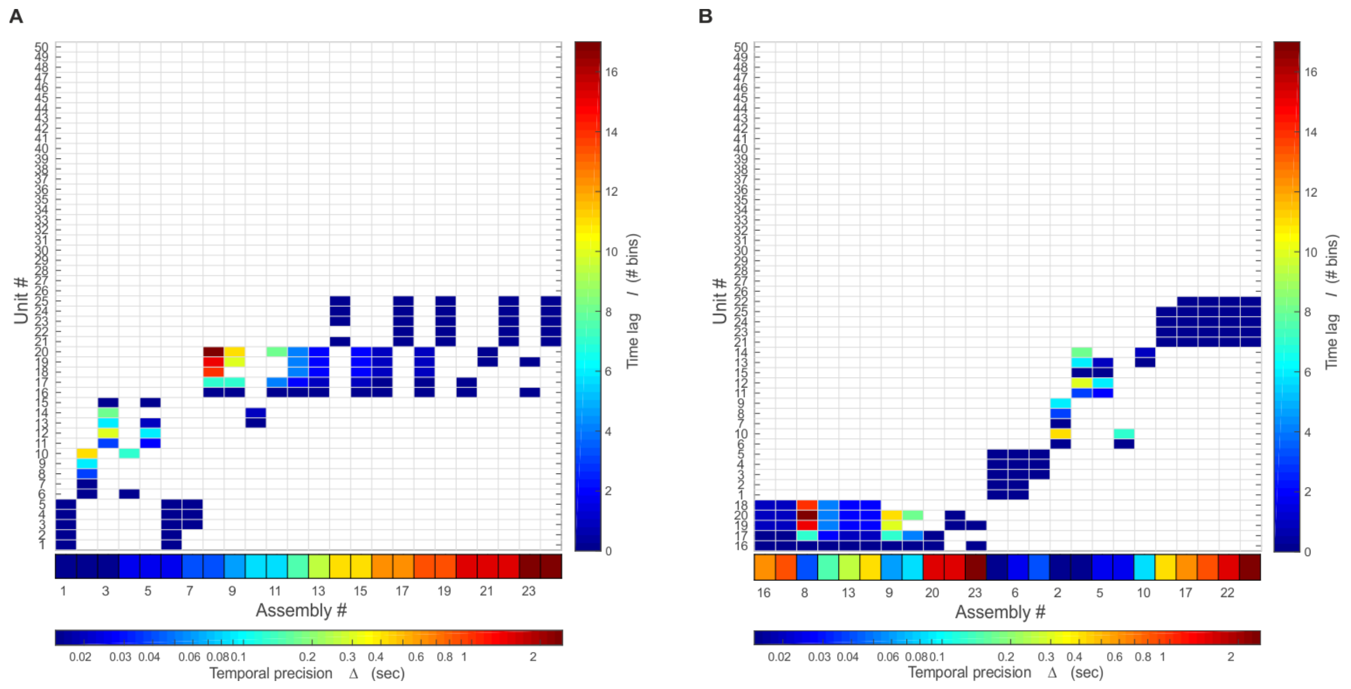

Figure T 2: Assembly assignment matrix. A) Display option: `display='raw'` (`display='ordunit'` produces the same figure as in this specific simulation units are already preordered); B) display option: `display='clustered'`.

Pruning options

From the assembly assignment matrix (Figure T2, cf. Fig. 1C) it is evident that the same assembly may be detected across a range of similar time scales, perhaps with fewer and varying assembly members at the *suboptimal* time scales. In such cases, for the sake of interpretability, it might then be convenient to prune the set of detected assemblies and keep only a subset of sufficiently distinct, most significant assemblies. The routine `pruning_across_bins.m` performs pruning of assemblies according to one of two criteria: `criteria = 'biggest'`, `criteria = 'distance'`. To obtain Figure 1 of the manuscript, as explained in the *Further assembly pruning* paragraph, we pruned the output of the `assemblies_across_bins` function with `criteria = 'biggest'` (Figure T3 - A):

```
criteria = 'biggest';
[As_across_bins_pr,As_across_bins_index_pr]=pruning_across_bins(As_across_b
ins,As_across_bins_index,nneu,criteria);

display='raw';
[Amatrix,Binvector,Unit_order,As_order]=assembly_assignment_matrix(As_acros
s_bins_pr, nneu, BinSizes, display);
```

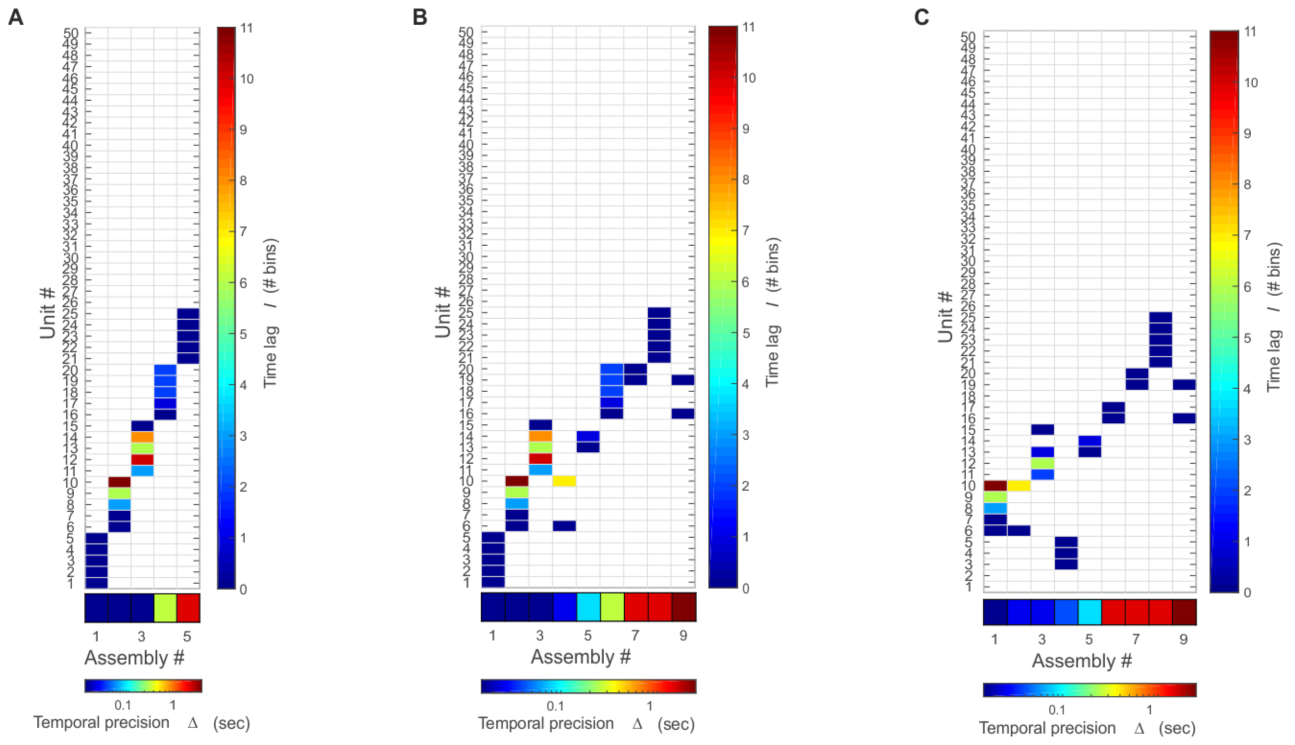

Figure T 3: Assembly assignment matrix after assembly pruning. A) Pruning option `criteria = 'biggest';` B) Pruning option `criteria = 'distance', style = 'pvalue'`; C) Pruning option `criteria = 'distance', style = 'occ'`. Both in B) and C): `th=0.3`.

Assembly activation time course

Information about when in time a specific assembly `i` is activated across the course of a recording session or task is contained in the structure variable `As_across_bins{i}.Time`, which records how often the full assembly spike pattern occurs within each time bin of the series (recall that this may be more than once for larger bins). Function `Assembly_activity_function` allows to specify further options for representing the assembly activity in time, namely `lagChoice`, which determines whether only the start time of an assembly or its full temporal extent is indicated, and `act_count`, which determines if and to which extent partial assembly activations are taken into account (i.e., with only some of the assembly units activated).

Full function description:

Main_assemblies_detection

Main function for detecting cell assembly patterns. It returns `assembly`, a structure containing information about all detected assemblies, their lag constellations and temporal scale.

Note: This function automatically saves variables in a file `assemblybin%.mat` which contains all assemblies detected for the bin widths specified in `BinSizes(%)`. All information in `assemblybin%.mat` is also returned in the output structure `assembly` (and hence this file may be removed in the end). However, if for some reason the algorithm is interrupted, `assemblybin%.mat` will have saved results for all bin widths already analyzed at that time.

Syntax

```
[assembly]=Main_assemblies_detection(spM,MaxLags,BinSizes)

[assembly]=Main_assemblies_detection(spM, MaxLags,BinSizes, ref_lag, alph, No_th,
O_th, bytelimit)

[assemblybin]=Main_assemblies_detection(spM,MaxLags,BinSizes,ref_lag)

[assemblybin]=Main_assemblies_detection(spM,MaxLags,BinSizes,[],alph)

[assemblybin]=Main_assemblies_detection(spM,MaxLags,BinSizes,[],[],No_th)

[assemblybin]=Main_assemblies_detection(spM,MaxLags,BinSizes,[],[],[],O_th)

[assemblybin]=Main_assemblies_detection(spM,MaxLags,BinSizes,[],[],[],[],bytelimit)
```

Function arguments:

`spM`	matrix with population spike trains; units run across rows and spike time stamps (not binned) across columns. Supernumerary matrix entries are to be filled with *NaN*.
`BinSizes`	vector of bin widths to be tested, interpreted in same units of measurement as used for `spM`.
`MaxLags`	vector of maximal lags to be tested. For bin width `BinSizes(i)`, all time lags between `-MaxLags(i)` and `MaxLags(i)` will be tested.

Optional parameters:

`ref_lag`	reference lag selection: if `ref_lag` >0 the reference lag used is fix and equal to `ref_lag`; if `ref_lag`<0 the reference lag for the pair $\#_{AB,l}$ is $-l$. Default =2;
`alph`	alpha level for statistical test; default = 0.05.
`No_th`	minimal number of occurrences required for an assembly to be accepted (all assemblies, even if significant, with less than `No_th` occurrences are discarded). Default = 0.
`O_th`	maximal assembly size (number of units). The algorithm will return assemblies composed by a maximum of `O_th` elements. Acceptable range = [2, `Inf`]. Default = `Inf`.
`bytelimit`	maximal size (in bytes) allocated for all assembly structures detected for a given bin widths. When the size limit is reached the algorithm stops adding new units. Default = `Inf`.

Function output:

`assembly.parameters`	parameters used to run `Main_assemblies_detection`.
`assembly.bin{i}`	contains information about assemblies detected with `BinSizes(i)`, probed with all lags `-MaxLags(i):MaxLags(i).`
`assembly.bin{i}.bin_edges`	bin edges (common to all assemblies in `assembly.bin{i}`).
`assembly.bin{i}.n{j}.elements`	vector of units composing the assembly (unit order corresponds to the order of agglomeration by the algorithm).
`assembly.bin{i}.n{j}.lag`	vector of time lags. `lag(z)` is the activation delay between `elements(z+1)` and `elements(1).`
`assembly.bin{i}.n{j}.pr`	vector of p-values. `pr(z)` is the p-value of the statistical test performed when adding `elements(z+1)` to the structure `elements(1:z).`
`assembly.bin{i}.n{j}.Time`	assembly activation time. It reports how often the full assembly occurs within each bin. `Time` always refers to the activation of the assembly unit listed first (`elements(1)`), which is not necessarily identical to the first assembly unit spiking.

`assembly.bin{i}.n{j}.Noccurrences`	number of assembly occurrences. `Noccurrences(z)` gives this number for the structure composed of units `elements(1:z+1)`.

Description of optional parameters

Reference lag
The user can chose if to use a fix or variable reference lag. When `ref_lag` >0 the algorithm uses `ref_lag` as fix reference lag (namely -`ref_lag` when *l*>0 and `ref_lag` when *l*<0), while when `ref_lag` <0 the algorithm uses as reference always *−l* (irrespectively of the actual `ref_lag` value). Default value `ref_lag`=2 as more conservative. See paragraph '*Choice of reference (correction) lag*' of the manuscript for theoretical discussion.

p-value
Alpha level used for statistical testing (Bonferroni corrections are performed automatically by the algorithm to adjust to this level). Default value: `alph=0.05`.

Threshold on the minimal number of assembly occurrences
Every time a new unit is found significant and added to a previously formed assembly, the number of assembly occurrences will usually drop (remaining the same only when the newly added unit spikes at *all* occurrences detected for the previously formed assembly). Therefore, depending on the study's aims, one might want to fix a minimum occurrence number below which no further units will be added.
Please note that this is not equivalent to discard in a second moment, after running the algorithm with `No_th=0`, all assemblies with less occurrences than `No_th`. In this case we would also discard all assemblies constituted by subsets of the undesired assembly, which could, as subset, have more occurrences than `No_th` (remember that the agglomeration process is designed to keep only maximal assemblies discarding all subsets fully included in a bigger one). Default value: `No_th=0`;

Threshold on the maximal assembly size (in units)
In some cases one might not be interested in detecting the full assembly structure, but only in subsets up to some size, for instance only in significant pairs (as for the analysis in Figure 4). The parameter `O_th` enables to set a maximal assembly size and thereby reduce computation time.

Threshold on the maximal assembly size (in bytes)
Another way to delimit the resulting file sizes and computation times is to fix the permitted `bytelimit` (default `=Inf`) for assemblies at a given time scale.

assemblies_across_bins

Function to rearrange assembly units in accordance with the order of their spiking times within the assembly pattern (implying that all lags will be >=0 in relation to the first assembly unit spiking). Assembly activation times are then redefined as the spike times of the earliest assembly unit. The function returns structure `As_across_bins` which contains all detected assemblies (ordered along temporal precision Δ).

Syntax

```
[As_across_bins,As_across_bins_index]=assemblies_across_bins(assembly
,BinSizes)
```

Function arguments:

`assembly`	output of `Main_assemblies_detection.m`
`BinSizes`	vector of bin widths to be tested.

Function output:

`As_across_bins`	structure containing all assemblies.
`As_across_bins{i}.elements`	vector of units composing assembly `i`, now reordered according to spiking times within assembly pattern.
`As_across_bins{i}.lag`	vector of time lags. `lag(z)` is the activation delay between `elements(z+1)` and `elements(1)`.
`As_across_bins{i}.Time`	assembly activation time. It reports how often the full assembly occurs within each bin. `Time` always refers to the activation of the assembly unit listed first (`elements(1)`), which is not necessarily identical to the first assembly unit spiking.
`As_across_bins{i}.pr`	vector of p-values. `pr(z)` is the p-value of the statistical test performed when adding `elements(z+1)` to the structure `elements(1:z)`.
`As_across_bins{i}.Noccurrences`	number of assembly occurrences.

	`Noccurrences(z)` gives this number for the structure composed of units `elements(1:z+1)`.
`As_across_bins{i}.bin`	bin width at which the assembly has been detected.
`As_across_bins_index`	original indices of reordered assembly units as referring to structure `assembly` returned by `Main_assemblies_detection.m`, that is reordered assembly `As_across_bins{i}` is the one given in `assembly.bin{As_across_bins_index{i}(1)}.n{As_across_bins_index{i}(2)}`.

assemblies_assignment_matrix

Function to display the assembly assignment matrix. It also returns the unit and assembly order which corresponds to the matrix rows and columns, respectively.

Syntax

```
[Amatrix,Binvector,Unit_order,As_order]=assembly_assignment_matrix(As
_across_bins,nneu,BinSizes,display)
```

Function arguments:

`As_across_bins`	structure returned by `assemblies_across_bins` (see above)
`nneu`	number of recorded units.
`BinSizes`	vector of bin widths tested.

Optional parameters:

`display='raw'`	assemblies and units are displayed as returned by `Main_assemblies_detection`, in the order of the temporal precision Δ associated with them.
`display='ordunit'`	units are rearranged in order to separate assembly from non-assembly units .
`display='clustered'`	assemblies and participating units are clustered such that assemblies with similar unit compositions are shown next to each other, but not necessarily in the order of their temporal precisions .

Function output:

`Amatrix`	`nneu` × `m` matrix (`m` = total number of assemblies). If unit `i` is an element of assembly `j`, `Amatrix(i,j)` contains the lag of `i` with respect to the first assembly unit.
`Binvector`	vector of bin widths. `Binvector(j)` gives the bin width at which `Amatrix(:,j)` has been detected.
`Unit_order`	order of units in `Amatrix`. `Unit_order(i)` gives the original index of the unit now entered in row `i` of `Amatrix`.

`As_vector`	order of assemblies in `Amatrix`. `As_vector(j)` gives the original index of the assembly now entered in column `j` of `Amatrix`.

pruning_across_bins

Function to prune the detected assembly set according to a variety of pruning criteria. It returns the same output as `assemblies_across_bins.m` but only for the assemblies retained after pruning.

Syntax

```
[As_across_bins_pr,As_across_bins_index_pr]=pruning_across_bins(As_ac
ross_bins,As_across_bins_index,nneu,criteria,th,style)
```

Function arguments:

`As_across_bins`			structure containing assembly information (output of `assemblies_across_bins.m`).
`As_across_bins_index`			information for linking assemblies in `As_across_bins` back to the structure `assembly (output of Main_assemblies_detection.m`).
`Nneu`			`# of recorded units.`
`criteria`	`= 'biggest'`		keeps only the largest assemblies, discarding all assemblies which are true subsets of larger assemblies in terms of their unit composition. If multiple assemblies are composed of exactly the same units, the one associated with the lowest p-value is maintained. This same type of pruning is already performed by the detection algorithm, but only among those assemblies detected *at the same temporal scale* (bin width).
`criteria`	`= 'distance'`		from all assemblies with a cosine distance smaller than `th` (in terms of shared elements, see sect. *Further assembly pruning*), only the most significant or the most frequent (see below) is retained.
	`= 'distance'`	`Th`	clustering threshold (cosine distance). It can assume values within the interval (0,1), where '0' implies no pruning and '1' would retain only one assembly. From each cluster only one assembly is kept.
	`= 'distance'`	`style = 'pvalue'`	Only the assembly with lowest p-value is kept.

`= 'distance'`	`style = 'occ'`	Only the assembly with the most occurrences is kept.

Function output:

`As_across_bins_pr`	same as `As_across_bins` but only with pruned assemblies
`As_across_bins_index_pr`	same as `As_across_bins_`index but for pruned assemblies.

Assembly_activity_function

This function returns the assembly activation profile across recording time according to different possible definitions.

Syntax

```
[assembly_activity]=Assembly_activity_function(As_across_bins,assembl
y,spM,BinSizes,lagChoice,act_count)
```

Function arguments:

`As_across_bins`	structure containing assembly information (output of `assemblies_across_bins.m`).	
`Assembly`	structure containing assembly information (output of `Main_assemblies_detection.m`).	
`spM`	Spike train matrix.	
`BinSizes`	vector of bin widths to be tested.	
`lagChoice =`	`'beginning'`	only start times of assemblies (i.e., the spiking times of the first unit in the pattern) are indicated.
	`'duration'`	Assembly activity is indicated for the full temporal extent of an assembly (from the first spike time of its composite units to the last spike in the pattern).
`act_count =`	`'full'`	only full assembly occurrences are considered (with all assembly units active).
	`'partial'`	partial activations are also taken into account (see details in the function descriptions below).
	`'combined'`	partial activations are weighted by the fraction of active assembly units (see details in the function descriptions below).

Function output:

`assembly_activity`	Each element `{i}` contains the activation (`assembly_activity{i}(:,2)`) in time (`assembly_activity{i}(:,1)`) of assembly `i`. Times refer to the bin centers.

Description of optional parameters

Assembly partial activations

As explained in the discussed in the reported example, the criteria to establish if an assembly is active or not in a determinate moment in time are various. In this function we give the possibility to choose among 3 options:

`act_count='full'`:

`assembly_activity` is computed adding the number of times the complete assembly pattern occurs in a bin.

`act_count='partial'`:

`assembly_activity` is computed adding the number of times the complete assembly pattern occurs in a bin plus fractions representing partial activation.

For example let's consider a synchronous assembly composed by N=4 units firing, in a specific bin t, respectively $n_t^1, n_t^2, n_t^3, n_t^4$ spikes. Let's define for each unit i a spike vector S_t^i (of dimension 1 x $max\{n_i\}$) containing a sorted collection of the unit spikes (for example, if $n_t^1 = 2$ and $max\{n_t^i\} = 4$ then $S_t^1 = [1, 1, 0, 0]$). The assembly activity at bin t is defined as:

$$activity(t) = \sum_{j=1}^{max\{n_t^i\}} \frac{\sum_{i=1}^{N} S_t^i(j)}{N}$$

note: While considering only full occurrences is a very strict criterion and looks as an unnecessary waste of information, the risk in using `act_count='partial'` is that the activity of a reduced number of assembly units overshadows the real assembly activity profile (e.g. if a unit pair AB has a common activity burst in a bin while the rest C and D of units assembly are silent, assembly activation will still returned as high).

`act_count='combined'`:

`assembly_activity` is computed adding the number of times the complete assembly pattern occurs in a bin plus fractions of partial activation weighted non linearly in order to penalize smaller assembly fractions.

In this case, the assembly activity at bin t is defined as:

$$activity(t) = \sum_{j=1}^{max\{n_t^i\}} \left(\frac{\sum_{i=1}^{N} S_t^i(j)}{N}\right)^5$$

FAQ

1. **What is the minimal number of units required for running the algorithm?**
   Two. The algorithm also works with just pairs given.

2. **What is the shortest-length spike train that could be analyzed?**
   The recording duration is in fact less important than the minimal number of bins required for decent statistics. A minimum of 100 bins is recommended (implying that only small Δs may be testable with short spike trains).

3. **How many spikes are necessary?**
   The Q_l statistics approximately follows an F-distribution. For this approximation to hold a minimum number of expected (by chance, that is under the H_0) coincident counts $\#_{AB}$ is required. In the algorithm, pairs of spike trains with $E[\#_{AB}] < 5$ are therefore automatically discarded. Essentially, this does not put a bound so much on the total number of spikes required, but more a bound on the temporal resolution that is feasible for a given set of spike trains. For instance, to achieve a temporal resolution of 10 ms for 2 units firing at 5 Hz, one should have about 20 sec of total recording time, or equivalently, about 100 spikes with each unit. Hence, in practice, with usually at least many minutes of total recording time, this condition should not be very limiting.

4. **If the assembly analysis takes too long or gets stuck, what should I do?**
   Change `O_th` or `bytelimit` options to reduce computational burden and use the automatically saved file `assemblybin%.mat` to recover already analyzed structure.