General Framework for Tracking Neural Activity Over Long-Term Extracellular Recordings

Fernando Julian Chaure[1] and Hernan Gonzalo Rey[2]

[1]Institute of Biomedical Engineering, University of Buenos Aires, Paseo Colon 850, Buenos Aires 1063, Argentina
[2]Department of Neuroscience, Psychology, and Behaviour, University of Leicester, Leicester, LE1 7HA,UK

February 11, 2020

Abstract

The recent advances in the chronic implantation of electrodes have allowed the collection of extracellular activity from neurons over long periods of time. To fully take advantage of these recordings, it is necessary to track single neurons continuously, particularly when their associated waveform changes with time. Multiple spike sorting algorithms can track drifting neurons but they do not perform well in conditions like a temporary increase in the noise level, sparsely firing neurons, and changes in the number of detectable neurons. In this work, we present Spikes_Link, a general framework to track neurons under these conditions. Spikes_Link can be implemented with different spike sorting algorithms, allowing the experimenter to use the algorithm best fitted to their recording setup. The main idea behind Spikes_Link is the blockwise analysis of the recording using overlapping sets of spikes to equally represent all the putative neurons being tracked on a given block. This way, we can link classes with clusters obtained in a new block based on an overlapping metric. Moreover, the algorithm can fix temporary sorting errors (splits and merges). We compared an implementation of Spikes_Link with other algorithms using long-term simulations and obtained superior performance in all the metrics. In general, the Spikes_Link framework could be used for other clustering problems with concept drift and class imbalance.

1 Introduction

The recording of extracellular activity from electrodes implanted in the brain is one of the most established techniques in contemporary neuroscience. In the past years, the chronic implantation of electrodes had allowed the collection of data over a long period of time. Analyzing this data generates new challenges. At first glance, the computational scalability with the amount of data is one of them (Carlson & Carin, 2019), but others related to the changes on the recording's properties over time are not fully characterized.

One of the main characteristics of a long recording is its stability; with a stable recording we can monitor the same neurons over a long period of time. In general, the stability of the recordings will depend on the electrodes used and the way they are anchored, leading to a large variety of scenarios, with examples such as tetrodes on the dorsal striatum of rats (Schmitzer-Torbert & Redish, 2004), high-density CMOS-integrated microelectrode array on mouse retina (Fiscella et al., 2012), immobile silicon probes in the mouse cortex (Okun et al., 2016), hippocampal multilayer electrode array in moving rats (Senkov et al., 2015), 672 microwires (in arrays with up to 128 wires) on the cortex of macaque monkeys (Nicolelis et al., 2003), independently movable arrays of nichrome electrodes on the macaque dorsolateral prefrontal cortex (Greenberg & Wilson, 2004), up to 1,024 polymer electrodes in freely behaving rats (Chung et al., 2019), Utah arrays into the human neocortex (Mégevand et al., 2017), depth electrodes in the human hippocampus that are anchored to the skull (Rey et al., 2015), and other recently developed neural recording electrode technologies (Hong & Lieber, 2019). Furthermore, other experimental factors could affect stability, for example, if the animal has its head fixed, if it is anesthetized, or freely moving.

To fully take advantage of the long-term recordings and study, for example, the variance of neuronal representations (Clopath et al., 2017) or the plasticity in neuronal processing (Lütcke et al., 2013), it is necessary to track neurons even when the stability fluctuates. A clear example of these issues can be found in recording sessions from the human medial temporal lobe where the microelectrodes are inserted inside a flexible probe that is anchored to the skull nearly 6 cm away from the recording site (Rey et al., 2015). In cases like this, similar responses to a given stimulus during sessions run on consecutive days that are associated to putative neurons with a different waveform (as shown in Fig. 1) could come from the same neuron following electrode drift, or from different neurons from the same assembly encoding the given stimulus. Tracking the single neuron activity throughout continuous recordings is the only way to discriminate these possibilities.

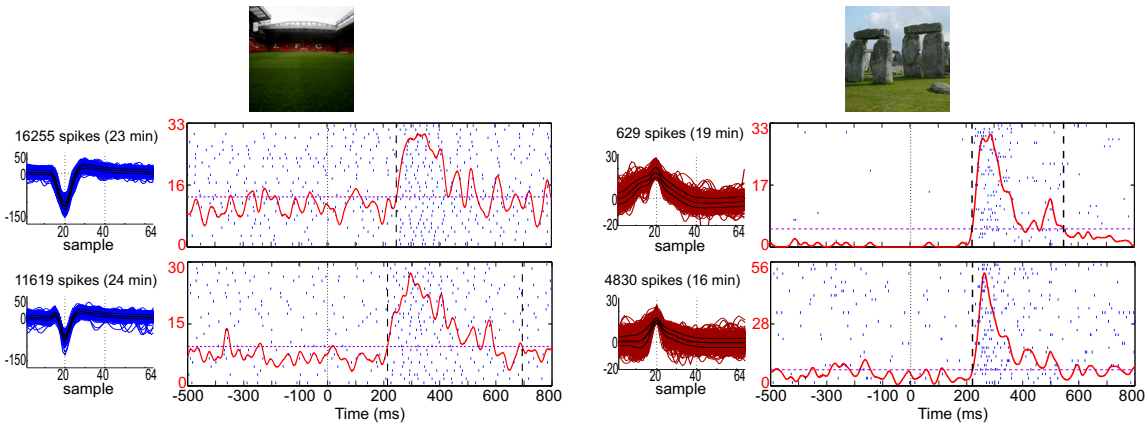

Figure 1: Raster plots from putative hippocampal neurons recorded in consecutive days from the same microwire. The left panel shows responses to Anfield (Liverpool FC stadium). Although the responses seem to be stable across days, the waveforms associated to these neurons are very different. Therefore, without tracking the neural activity over the whole time between sessions, it would be hard to argue whether it is the same neuron or a different one from the network encoding the concept "Anfield". The right panel shows responses to Stonehenge on consecutive days. In this case, a sparse neuron in the first session leads to a very selective response. In the next session, the response seems to be maintained, but its selectivity is not the same. Tracking the activity would allow us to distinguish between a sorting error (the cluster associated to the neuron being contaminated by spikes from other neurons) and an actual change in the neuron's firing dynamics.

There have been recent work to develop methods for long-term tracking of neurons over recordings sessions (Fraser & Schwartz, 2012; Tolias et al., 2007; Emondi et al., 2004; Eleryan et al., 2014; Dickey et al., 2009), over consecutive blocks of spikes or time (Niediek et al., 2016; Bar-Hillel et al., 2006; Wolf & Burdick, 2009; Shalchyan & Farina, 2014; Shan et al., 2017; Dhawale et al., 2017), or gradually over the whole recording (Calabrese & Paninski, 2011; Pouzat et al., 2004; Franke et al., 2009). All these approaches can handle different degrees of recording stability, but in general their performance is very poor when tracking neurons during epochs with high noise levels, sparse-firing neurons, or a variable number of detectable neurons (neurons appearing or disappearing).

To handle these difficulties, we present a general framework to track drifting neurons over long periods of time that does not rely on a specific spike sorting approach, i.e. the user can choose the spike sorter that will be used in the core of the tracking algorithm (Chaure et al., 2018; Yger et al., 2018; Jun et al., 2017; Chung et al., 2017; Hilgen et al., 2017).

2 Method

The method presented here, that will be called *Spikes_Link*, requires an initial detection of spikes and their separation in consecutive blocks of $N_{\max} = 20,000$ spikes. When the length of a resulting block

exceeds $T_{max} = 30$ minutes, a smaller 30-minutes block is selected as long as the number of spikes is larger than $N_{\min} = 10,000$. Otherwise, a block with N_{min} spikes is selected. Still, it would be possible to adapt the algorithm to support blocks defined by a given duration instead of the number of spikes, where spike detection could be done within each block by the specific sorter chosen.

Each block with N_{t_i} spikes will be labelled t_i with i=1,2...; given the variability of the firing rates the duration of each block will be, in general, variable. These blocks will be sequentially examined to track the previously detected putative neurons. As we track the activity of a putative neuron that is associated to class j, we define $C_j^{t_i}$ as the collection of spikes up to t_i that were assigned to class j. At the same time, when sorting is performed on block t_i, spikes are separated into clusters $\tilde{C}_k^{t_i}$. To track the neurons' activity, the spikes from each cluster will be assigned to one of the available classes; although they could also be assigned to a new class if necessary, or even discarded.

2.1 Tracking Metric

On a given block t_i, a set of $N_{OS} = 500$ overlapping samples (spikes) is selected for each available class j, which is denoted as $OS(C_j^{t_i})$. To create each set, we first choose the most recent $N_{OS}/(1-p_{out})$ samples, and remove the proportion $p_{out} = 0.2$ of the spikes with the largest euclidean distance from the median waveform, as they could be seen as potential outliers. When less than N_{OS} spikes are available, the "synthetic minority over-sampling technique" (Chawla et al., 2002) is used to complete the set. On the following block t_{i+1}, the spikes from all the overlapping sets $OS(C_j^{t_i})$ are added to the $N_{t_{i+1}}$spikes before applying the sorting algorithm to that block. Figure 2 depicts the procedure for creating the overlapping sets on consecutive blocks with two drifting classes.

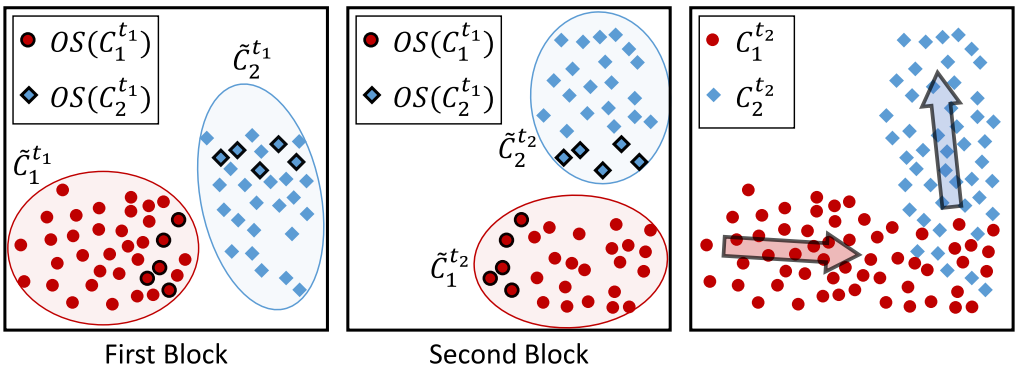

Figure 2: **Schematic example with two drifting classes and the use of the overlapping sets to track them**. In the left panel, two clusters are detected by a sorting algorithm, and assigned to classes $C_1^{t_1}$and $C_2^{t_1}$. Then, an OS is created for each of these classes, i.e. $OS(C_1^{t_1})$ and $OS(C_2^{t_1})$. In the middle panel, the spike sorting result on the next block is shown, where clusters $\tilde{C}_1^{t_2}$ and $\tilde{C}_2^{t_2}$ were separated. Since the overlapping sets were perfectly separated between the clusters, the overlapping values will be 1 (see equation 1), and clusters $\tilde{C}_1^{t_2}$ and $\tilde{C}_2^{t_2}$ can be easily included to the classes $C_1^{t_1}$ and $C_2^{t_1}$, resulting in the tracked classes $C_1^{t_2}$ and $C_2^{t_2}$, as shown in the right panel (the arrows represent the drifting direction of each class). In general, the feature space on each block would be different, as it is defined by the sorting algorithm on each block.

Based on the overlapping sets, we can define a metric for tracking classes between consecutive blocks. Specifically, when processing block t_i we can compute the proportion of spikes from each overlapping set $OS(C_j^{t_{i-1}})$ that can be found on each cluster $\tilde{C}_k^{t_i}$ th, i.e.:

$$Ov_{j,k}^{t_i} = \frac{\left|OS(C_j^{t_{i-1}}) \cap \tilde{C}_k^{t_i}\right|}{N_{OS}}, \tag{1}$$

where $|\cdot|$ denotes the number of elements in the set. This "overlapping value" lies between 0 and 1, with large values providing strong evidence that the new spikes in cluster $\tilde{C}_k^{t_i}$(i.e. the ones resulting following

the removal of those from the overlapping sets) should be incorporated to the class $C_j^{t_i}$. From here on, when we say that spikes from a given cluster should be incorporated to a given class, we always assume that all spikes from the overlapping sets have been removed in advance. To remove artifacts, only the clusters with a firing rate higher than 0.03 Hz (computed after removing the spikes from the overlapping set) will be considered for tracking on each block; otherwise, these spikes are discarded.

2.2 Tracking Loop

The tracking loop is applied after a sorting algorithm classified the spikes of a given block t_i, which includes all the sets OS associated to the classes from the previous block t_{i-1}. Following the calculation of the overlapping value $Ov_{j,k}^{t_i}$ for every pair j, k, each cluster $\tilde{C}_k^{t_i}$ will be associated to one of the following cases:

1. The cluster is labeled as a "match" if it has an overlapping value higher than 0.5 against a single class j^* from the previous block. In this case, all the spikes from the cluster are incorporated into the class j^* (see Fig. 3A).
2. The cluster is labeled as a "split" if its highest overlapping value against all classes is between 0.2 and 0.5, and the second highest value is less than 50% of the highest and lower than 0.2 (see cluster $\tilde{C}_1^{t_2}$ in Fig. 3B).
3. The cluster is labeled as a "merger" if it has overlapping values higher than 0.5 with more than one class. In this case, the procedure explained below in 2.4 is applied (see Fig. 3D).
4. In any other case, the cluster is labeled as a "new class" (see cluster $\tilde{C}_3^{t_2}$ in Fig. 3C).

If the sum of the overlapping values from a given class j' for all the clusters labeled as split or match is higher than 0.5, the spikes from the split clusters are incorporated to the class j' (see clusters $\tilde{C}_1^{t_2}$ and $\tilde{C}_3^{t_2}$ in Fig. 3B); otherwise they are assigned to a "new class" (see cluster $\tilde{C}_1^{t_2}$ in Fig. 3C). Using this criterion it is possible to fix cases where the sorting algorithm generates overclustering on a given block.

2.3 Class Survival

When dealing with real data, it is expected that some blocks could be dominated by noise or artifacts, so the contribution of neurons could be masked. To handle this issue, a survival criterion was added to the method. Under this condition, any class would survive for a maximum of $CS = 3$ consecutive blocks, even if the tracking method is not able to associate clusters to that class in the intervening blocks. Within these blocks, the same overlapping set for the class is used (as no new spikes are incorporated). If no spikes are assigned to the class after CS consecutive blocks, the class is labeled as "disappeared", and it is no longer tracked.

2.4 Merge Criteria

When a merger is detected for a cluster $\tilde{C}_k^{t_i}$, a set of steps is followed to protect the classes against sorting errors in particular blocks. First, if more than one of the classes with $Ov_{j,k}^{t_i} > 0.5$ were "new classes" in block t_{i-1}, they are merged into a single "new class" in block t_{i-1}; if this results in just a single class associated to the merger, that class is labeled as a "new class" and matched with $\tilde{C}_k^{t_i}$. Next, if there is only one "new class" and only one "old class" associated to the merger, these classes are merged as a single "old class" in block t_{i-1} and matched with $\tilde{C}_k^{t_i}$.

In general, the cluster labeled as a merger will have a set of classes from the previous block associated to it, which we call the "merge list" ML. The first time the merger cluster is detected, it is actually considered as a temporary merger (see Fig. 3D left). In this scenario, the overlapping sets associated to the classes from the list ML will be included in the next block, to give the algorithm a chance to find them separately and continue the individual track of the putative neurons (see Fig. 3D middle). This procedure is repeated for $MS = 3$ consecutive blocks as long as the detected merger is associated to

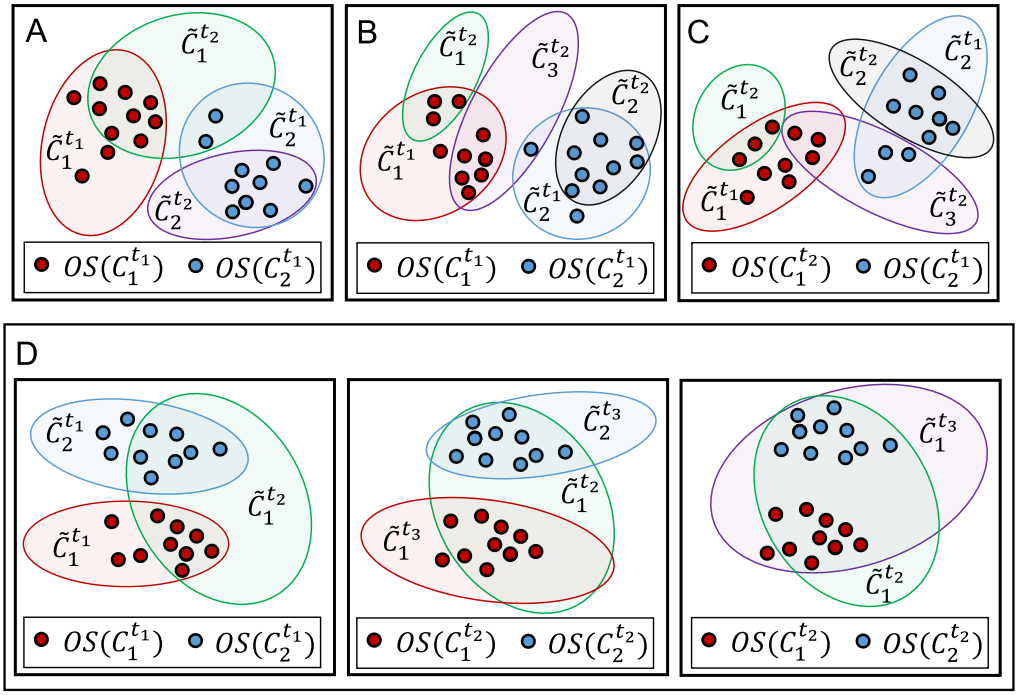

Figure 3: **Schematic examples of different tracking scenarios.** For simplicity, only the elements of the overlapping sets are shown for each cluster. **A** Clusters $\tilde{C}_1^{t_2}$ and $\tilde{C}_2^{t_2}$ generate a match with classes $C_1^{t_1}$and $C_2^{t_1}$as the overlapping sets lead to large overlapping values. **B** In this case, clusters $\tilde{C}_2^{t_2}$ and $\tilde{C}_3^{t_2}$ are matches with classes $C_2^{t_1}$and $C_1^{t_1}$, respectively, but cluster $\tilde{C}_1^{t_2}$ is labeled as a split with respect to class $C_1^{t_1}$. Since clusters $\tilde{C}_3^{t_2}$ and $\tilde{C}_1^{t_2}$ together include the majority of the spikes of $OS(C_1^{t_1})$, they are automatically combined and included in class $C_1^{t_1}$ to generate class $C_1^{t_2}$. **C** For this example, $\tilde{C}_2^{t_2}$ is a match with class $C_2^{t_1}$, but clusters $\tilde{C}_1^{t_2}$ and $\tilde{C}_3^{t_2}$ are labeled as new classes. Cluster $\tilde{C}_1^{t_2}$ is a split of class $C_1^{t_1}$, but without another split or match it cannot be included in the class. Cluster $\tilde{C}_3^{t_2}$ has a reasonable overlap with both classes (so it cannot be a split) but not large enough to generate matches. **D** In the left panel, cluster $\tilde{C}_1^{t_2}$ is labeled as a temporary merger at block t_2 (associated to classes $C_1^{t_1}$and $C_2^{t_1}$, that define its ML list). In the next block, the same overlapping sets will be used (i.e. $OS(C_1^{t_2}) = OS(C_1^{t_1})$ and $OS(C_2^{t_2}) = OS(C_2^{t_1})$). The middle panel shows that during block t_3, clusters $\tilde{C}_1^{t_3}$ and $\tilde{C}_2^{t_3}$ were matched to the previous classes, so the merger is not confirmed and cluster $\tilde{C}_1^{t_2}$ will be later resolved as explained below in the Graph Reduction phase (see Fig. 4D). Alternatively, as shown in the right panel, cluster $\tilde{C}_1^{t_3}$ is another merger associated to the same ML list. If this situation is repeated for MS consecutive blocks, the merger is confirmed, a new class is associated to the merged cluster in all MS blocks, and the individual classes from the ML list "disappear".

the same ML list (see Fig. 3D right). If the situation persisted for all MS blocks, the merger is then confirmed, a new class is associated to the merger cluster in all MS blocks, and the individual classes from the ML list "disappear", i.e. they are no longer used to create overlapping sets in the following blocks and will therefore be no longer tracked individually.

2.5 Graph Reduction

After the analysis of a long-term recording has been performed, some classes will be present in only a few blocks, and later disappear or be merged with others. Usually, these classes are non-resolved overclusterings from another class within a block and, in most cases, they should be combined together. At the same time, some of them can be created by errors from the spike sorting algorithm and should be discarded to preserve the quality of the results. We define as spurious classes (SC) those that appeared in less than $S_{TH} = 3$ blocks throughout the whole recording, and the clusters associated to these classes

in those blocks are not labeled as mergers. Otherwise, if a class appears in at least S_{TH} blocks, it is considered a "stable class".

First, a graph is created for the whole recording, where nodes that could be mergers, new classes, or matches, are connected with edges across consecutive blocks according to the tracking criteria explained above (see Fig. 4A). Next, all the graph components (isolated sub-graphs) are obtained; in the case of Fig. 4A , 3 sub-graphs ***a-b***, ***c-d***, and ***e***. If no "stable classes" are included on a given sub-graph, all the classes in that sub-graph are discarded (see black node in sub-graph ***e*** in Fig. 4A). Then, minimal sub-graphs containing spurious classes are further isolated by removing edges linking nodes from the same stable classes (i.e. edges connecting nodes of the same color). For example, in Fig. 4A , these sub-graphs can be found within the sub-graphs ***a***, ***b*** and ***c***, with resulting sub-graph ***d*** having no spurious classes.

All the minimal sub-graphs containing SC are then analyzed with the following criteria:

1. If only one stable class is on the sub-graph, all the SC are combined into that stable class. For example, in Fig. 4A red and green are combined in t_{i+5}, while yellow, brown and grey are combined in t_{i+3}.
2. If only two stable classes are in the sub-graph, denoted by C^* and C', with C^* being a merger and C' being on its ML list, then all the spurious classes in the sub-graph are combined with C' (see sub-graph ***a*** in Fig. 4A, where green would be C^*, orange would be C' and blue would be the spurious class).
3. In any other case, the SC are discarded.

The changes in the spurious classes will modify the composition of the mergers. If the updated ML list has a single class, then the merger class can be seen as a match of that class (see sub-graph reduction ***a***, ***b*** and ***c*** in Fig. 4B). This step (merger reduction) might be applied recursively until all the ML lists can no longer be updated. Finally, stable classes throughout the recording might be temporarily merged (e.g. yellow and violet at t_{i+1}) or permanently merged after a certain block (e.g. yellow and violet after t_{i+8}). In those cases, the nodes are labeled in a way that represents that those classes are merged at those blocks (Fig. 4B, ***d***). The classes should be handled with care during these blocks; e.g. it would make no sense to compute a cross-correlation between them.

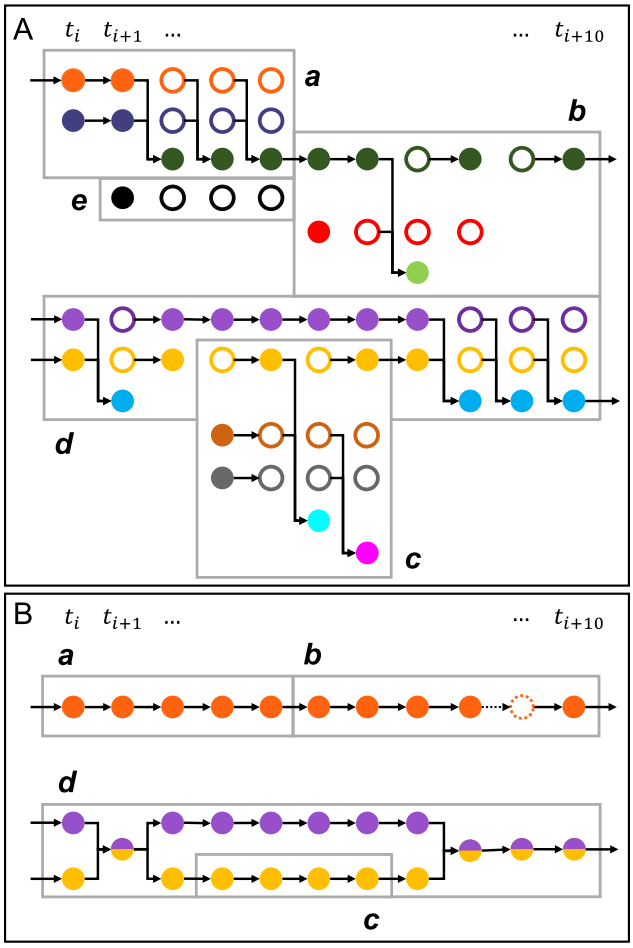

Figure 4: **Example of different scenarios of graph reduction on 11 consecutive blocks. A** For a given class, multiple nodes are created, one for each block from the creation of the class until its possible disappearance. Filled circles are nodes associated to classes that were updated on that block with new spikes. The edges of the graph connect the nodes across different blocks according to the rules explained above in the tracking loop. Empty circles with a solid line denote classes that were not updated with new spikes on that block, although their overlapping sets were included in the sorter. Different classes are represented by different colors. The arrows at the very left of the graph denote that the class has been tracked over several previous blocks, a class with no arrow on the left is a new class, and if a class ends with an arrow at the very right of the graph it means that it will continue being tracked afterward. Sub-graph ***f*** has no stable classes so it will be discarded. In ***a*** the blue node is associated to a new class at t_i and matched in the next block; next, it merged with the stable orange class, leading to the green nodes, with the merge being confirmed after 3 blocks. This behavior could be the result of overclustering (blue class) by the sorting algorithm. In ***b*** the green class has a temporary overclustering (red class), that was not automatically merged in the following block, but was merged in the next one (light green node), but only for that block. In ***c***, multiple SC (grey and brown) are created (possibly due to overclustering), but they are not all merged together on the next block. Only partial temporal merges with other classes were found, without sharing the exact same *ML* list. The case ***d*** depicts a temporary merge during only one block, which could be easily caused by changes in the noise level of the recording. After a few blocks of separable activity, the classes were then combined in a confirmed merger. **B** Resulting graphs following the graph reduction criteria. In this case, 3 different classes are obtained during the 11 blocks of recording analyzed. Case ***a*** uses criterion 2, whereas ***b*** uses criterion 1; both, in turn, are combined into a single class following merger reduction. Notice that the class was not found during block t_{i+9} (dotted circle), but the survival criterion prevented it from disappearing. Case ***c*** is combined all the classes into the yellow using criterion 2, and using merge reduction in conjunction with case ***d***, leads to 2 classes being tracked.

3 Results

3.1 Simulated Dataset

The simulated dataset was created using 15 simulations with 4 to 6 neurons from (Pedreira et al., 2012). This dataset has been made public (Rey et al., 2015), and it has been used for testing the tracking ability of spike sorting algorithms (Niediek et al., 2016). On each simulation we concatenated 70 repetitions of the simulated spikes and linearly increase/decrease the mean waveform amplitude of each class by a factor of 2.5 (i.e. increase from 1 to 2.5, or decrease from 2.5 to 1), but without changing the distance between each spike an its corresponding mean waveform (i.e. each spike gets the old mean waveform subtracted and then the new mean waveform added, with the latter being time-dependent). This way, we changed the centroid of the cluster associated with the class without affecting the high order statistics. Finally a Gaussian noise was added to each spike with a standard deviation of 10% of the maximum amplitude of the waveforms associated to the original classes (i.e. before drifting). A similar approach was used by Niediek et al. (Niediek et al., 2016), but all the spikes were just scaled throughout the simulation to simulate the drifting, so higher order statistics were affected. An example of one of our simulations is shown in Fig. 5.

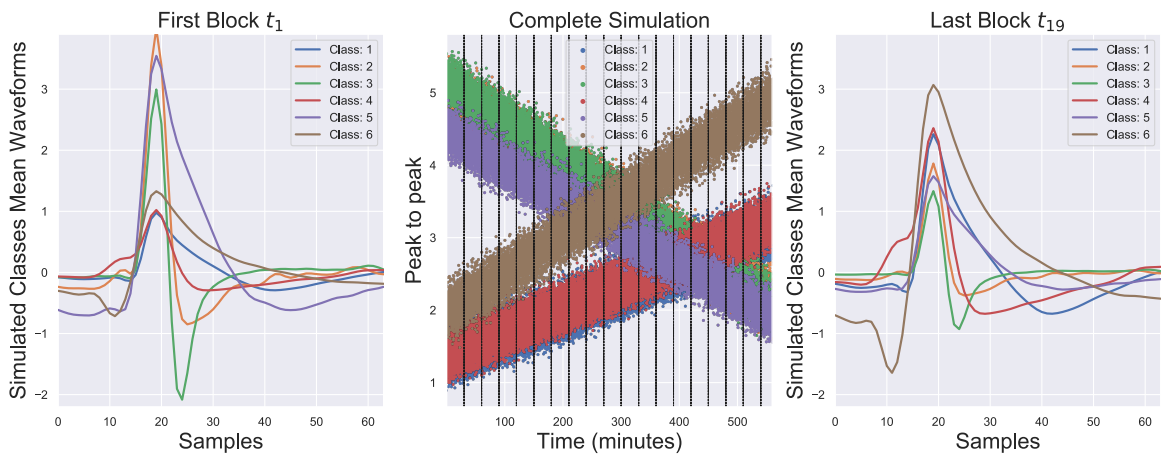

Figure 5: **Example of simulated long-term recording number 24**. The spikes were separated in blocks as explained in the Method section (19 blocks in this example). The mean waveforms of the simulated neurons (classes) in the first block are shown in the left panel. In the middle panel the peak to peak amplitude of each spike is shown color-coded, so the drifting dynamics of each class can be seen. The vertical lines indicate the separation between blocks. Finally, the right panel shows the mean waveforms of the neurons during the last block. In this simulation, half of the neurons increased their amplitude and the other half decreased it.

3.2 Performance evaluation

For the implementation of Spikes_Link presented here, we used Wave_Clus (Chaure et al., 2018) as the spike sorting algorithm applied on each block. In the following, we labeled this combination as Spikes_Link_WC. We compared the performance of Spikes_Link_WC against the standard Wave_Clus (i.e. providing the whole long-term recording to the spike sorting algorithm) and Combinato (Niediek et al., 2016), a spike sorting algorithm that can be applied to long-term single channel recordings and uses a specific approach to solve the drifting problem. In both cases, the default parameters presented in the respective papers were used (in the case of Combinato, other sets of parameters were tested but without qualitative changes in the performance).

We used different metrics to quantify the performance. Recall, precision and accuracy were calculated

in the same way as defined by Jun et al. (Jun et al., 2017). Given the set of spikes from a simulated neuron k, and a class c from a given algorithm (i.e. a putative single neuron), $n^{k,c}_{match}$ is defined as the number of spikes that belong to the intersection of both sets, $n^{k,c}_{fp}$ is defined as the number of spikes belonging to class c that are not from the neuron k, and $n^{k,c}_{miss}$ is defined as the number of spikes from neuron k that are not included in class c. Then, the metrics are defined as:

$$precision\,(k,c) \;=\; \frac{n^{k,c}_{match}}{n^{k,c}_{match}+n^{k,c}_{fp}}$$

$$recall\,(k,c) \;=\; \frac{n^{k,c}_{match}}{n^{k,c}_{miss}+n^{k,c}_{match}}$$

$$accuracy\,(k,c) \;=\; \frac{n^{k,c}_{match}}{n^{k,c}_{match}+n^{k,c}_{miss}+n^{k,c}_{fp}}$$

Finally, for each neuron k the class $c*$ with the highest accuracy is used as the best match, and the metrics for the pair (k,$c*$) are the ones reported for that neuron.

Figure 6 shows a simulated long-term recording with 4 neurons (top) and the results obtained by each algorithm. The borders of each block calculated by Spikes_Link are shown with dotted lines, and the mean waveform of the classes within each block was computed. Spikes_Link_WC was able to track the classes with high performance (μ_a=0.998, μ_p=0.99, and μ_r=0.99, for accuracy, precision, and recall), but the other algorithms (particularly the standard Wave_clus) tend to split the simulated neurons in multiple classes (overclustering). In this simulation, Wave_clus did not merge parts of the simulated neurons (as reflected by its precision with μ_p=0.99), but Combinato merged together two simulated neurons (as reflected by its precision with μ_p=0.75). This is due to the final step of Combinato, which merges together all the classes with mean waveforms that are close enough in euclidean space without using temporal information.

The performance for all the simulated neurons (each represented by a symbol with a unique combination of color and shape) is shown for each algorithm in Figure 7. The accuracy of Spikes_Link_WC was significantly larger than the one of the other algorithms ($p<2e^{-19}$). This difference is produced mainly by the lower recall obtained for the other algorithms ($p<1e^{-15}$). A lower recall value is expected when the algorithms do overclustering. On the other hand for some simulated neurons, the performance showed a lower precision, as different neurons were merged into the same class. Still, this was resolved better by Spikes_Link_WC , showing a significant improvement in precision against the other algorithms ($p<2e^{-3}$) .

Additional comparisons were performed. To measure the number of extra classes that a method creates for a given simulation, a metric called 'Additional Classes' was defined as the total number of classes in the solution minus the number of clusters labeled as the best match of neurons in the simulation (to avoid negative values if an algorithm merges two neurons). A large value of this quantity could be an important limitation for long-term recordings because the final number of classes to analyze, curate and compare will grow with the duration of the recording. On the other hand, to measure the general agreement of the sorting with the ground truth, we used the Adjusted Rand Index (Steinley, 2004). Figure 8 shows the results, where Spikes_link_WC always obtained a better agreement (higher Adjusted Rand Index) with the ground truth ($p<7e^{-5}$) and a significantly lower number of additional classes ($p<4e^{-3}$). These results support the idea that the design choices of the Spikes_link framework can improve spike sorting algorithms during long-term recordings with possible nonstationarities(e.g. under drifting conditions).

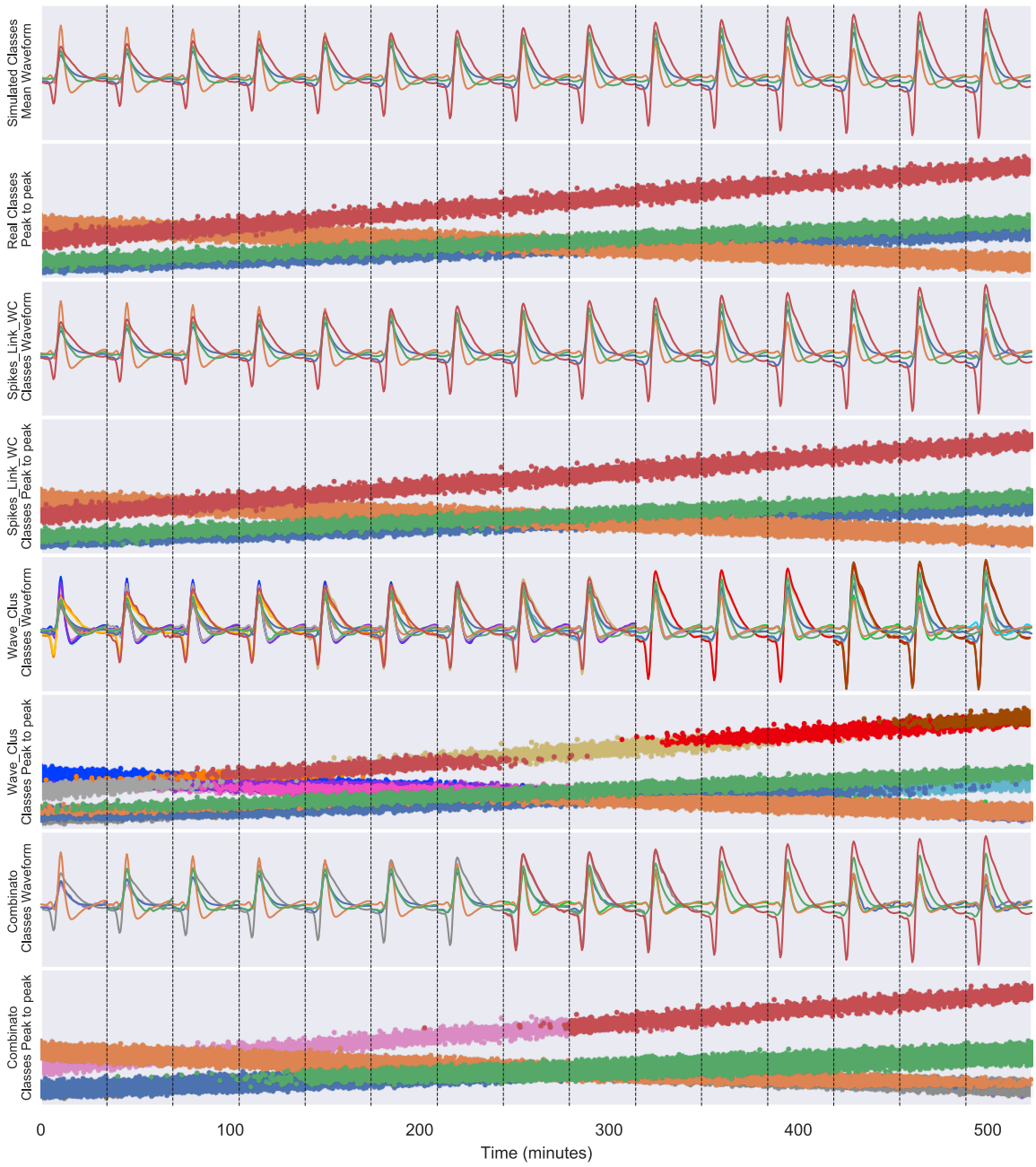

Figure 6: **Example of results of the simulated recording number 21.** The mean waveforms of the classes and the peak to peak amplitude of their spikes are shown for the ground truth (top) and each algorithm. The dotted vertical lines delimit the segments that are used by Spikes_Link as blocks; the mean waveforms for each class are also calculated within these segments. For each algorithm we computed the mean value of the accuracy (μ_a), recall (μ_r) and precision (μ_p), and their range ($[min, max]$) across all the simulated classes. Accuracy of each algorithm; Spikes_Link_WC: μ_a=0.998, [0.99, 1]; Wave_clus: μ_a=0.52, [0.26, 0.83]; Combinato: μ_a=0.49, [0.21, 0.73]. Recall of each algorithm; Spikes_Link_WC: μ_r=0.99, [0.996, 1]; Wave_clus: μ_r=0.53, [0.26, 0..83]; Combinato: μ_r=0.66, [0.5, 0.76]. Precision of each algorithm; Spikes_Link_WC: μ_p=0.99, [0.997, 1]; Wave_clus: μ_p=0.99, [0.97, 1]; Combinato: μ_p=0.75, [0.23, 1]. In this example, Spikes_Link_WC tracked the drifting classes correctly, Wave_clus generated a strong overclustering, and Combinato also presented overclustering but merging two neurons together as well. The other metrics are shown in Fig. 8.

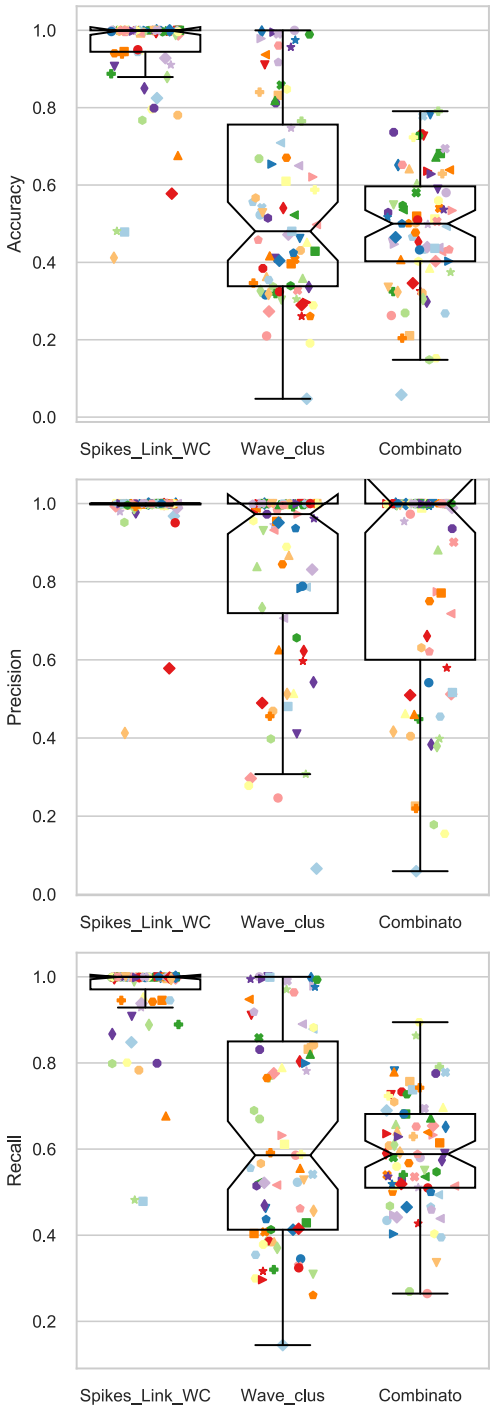

Figure 7: **Performance per simulated neuron.** This figure shows the performance obtained from each simulated neuron using the compared algorithms. Box-and-whisker plots were constructed for each algorithm and metric to summarize their results. Paired sign tests were used to evaluate the differences in performance between Spikes_link_WC and the other methods. Spikes_link_WC showed better results in all cases; accuracy led to $p<2e^{-19}$; precision led to $p<2e^{-3}$, and recall led to $p<1e^{-15}$.

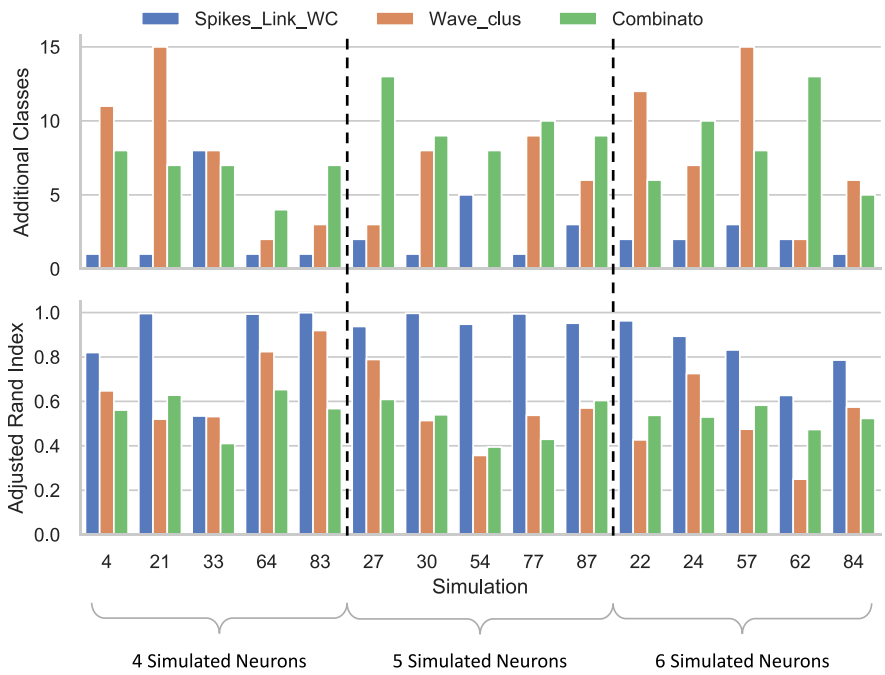

Figure 8: **Clustering performance per simulation.** On the top panel, the number of Additional Classes was calculated for each simulation and algorithm. Computing a paired sign test for this metric, Spikes_link_WC obtained $p<4e^{-3}$ against the other algorithms. On the bottom panel, the Adjusted Rand Index measures the similarity between the ground truth and the spike sorting output (i.e. the class label assigned to each spike). A paired sign test was used to evaluate the differences between Spikes_link_WC and the other methods, leading to $p<7e^{-5}$.

4 Discussion

The simplest approach that has been used to decide if clusters from different recording sessions could be associated to the same neurons is based on comparing their waveforms and other metrics (e.g. mean firing rate) and link the clusters that conserve similar proprieties (Tolias et al., 2007; Fraser & Schwartz, 2012; Emondi et al., 2004; Eleryan et al., 2014; Dickey et al., 2009). This approach requires the recording to be stable enough so that the waveforms and the firing dynamics of the neurons do not change significantly across sessions. The main advantage of this simple cluster-comparison approach is that any spike sorting algorithm could be used to obtain the clusters on each session.

The application of a Bayesian framework to model blocks of spikes and then apply previous information and transition probabilities to track the neurons can be used to follow gradual changes across the recording (Bar-Hillel et al., 2006; Wolf & Burdick, 2009; Shalchyan & Farina, 2014; Shan et al., 2017). However, some of these methods require a fixed number of neurons across the recording (Shalchyan & Farina, 2014; Shan et al., 2017) and others use algorithms with high computational cost to handle a variable number of neurons (Wolf & Burdick, 2009; Bar-Hillel et al., 2006). The approach has the advantage of dividing the tracking procedure in two smaller problems: a clustering on each stable block and the calculation of the transition probabilities.

Other methods allow neurons to gradually drift from an initial analyzed segment (Franke et al., 2009; Calabrese & Paninski, 2011; Pouzat et al., 2004), but they are sensitive to errors produced in the sorting of the initial segment and the change of the number of neurons throughout the recording. These methods remark the necessity of an error-correcting approach with enough flexibility to update the number of neurons being tracked. A method called FAST (Dhawale et al., 2017) is a clear example that it is possible to implement such a method, but its high computational cost and lack of intuition on the effect of some of its parameters and how to control them makes it difficult to use this method in practice.

Some well-known spike sorting methods can handle drifting waveforms (Chung et al., 2017; Jun et al., 2017; Niediek et al., 2016), yet it is possible that, for long-term recordings, all the spikes with a similar waveform will be assigned to the same cluster without taking in consideration the time difference between them.

We combined all these observations and developed Spikes_Link, a general framework to fully track neurons and handle spike sorting errors and stability fluctuations using ad hoc rules. The method requires a certain spike sorting algorithm (the one preferred by the user) to sort a block of spikes and then includes on the next block subsets of the spikes of the classes being tracked. Based on these sets, we introduced a metric between clusters from a given block and classes being tracked from previous blocks. As all the classes are equally represented in the sets, this provides an advantage for tracking sparsely-firing neurons. In addition, after analyzing all the available blocks, we implement a graph reduction step to reduce the number of spurious classes.

The metric and the relationship between clusters in Spikes_Link have some similarities to the MONIC framework (Spiliopoulou et al., 2006) for modeling and tracking general cluster transitions. However, MONIC tracks the dynamics of classes without considering the possibility of clustering errors, and uses overlapping time windows instead of sets as we do, which would affect tracking over sparse neurons. Finally, the framework presented in this work can be applied to other problems, such as data streams with concept drift and class imbalance (Hoens et al., 2012).

The main parameters used by Spikes_Link are:

- N_{min}: minimum number of spikes recommended for a given block (it could depend on the spike sorting algorithm chosen).
- N_{max}: maximum number of spikes recommended for a given block.
- T_{max}: maximum period of time where the recording can be presumed stable.
- N_{OS}: number of samples per class in the overlapping set.
- p_{out}: proportion of spikes that would be discarded as potential outliers to create the overlapping set.
- MS: number of consecutive blocks for a temporal merger before it is confirmed.

- CS: number of consecutive blocks that a class can survive without a matched cluster. Note the constraint $MS \leq CS$.
- S_{TH}: minimum amount of blocks with spikes for a non-spurious class.

Some parameters can depend on the specific spike sorting algorithm of choice (N_{min}, N_{max}) or the expected quality of the clustering (p_{out}). An initial expected stability is parameterized by T_{max}, with some drifting being allowed within a block without compromising the performance (as seen in Fig. 6), although this might depend on the chosen spike sorting algorithm. In this work, N_{OS} was set to 500; a much larger value would increase the computing time used by the spike sorting algorithm, whereas a much smaller value would affect the performance of the overlapping metric. For sparse neurons that could be absent in some blocks, the parameter CS will cap the number of blocks in which the same waveform will be searched. MS measures how many mergers of previous classes need to be detected before loss of isolation is confirmed. This is also important, as blocks with high levels of noise can affect the performance on a given block, and having $MS > 1$ gives the chance to the algorithm to continue isolating individual clusters in the following blocks. Finally, S_{TH} defines the minimum scale in blocks at which a class is considered as valuable. Therefore, the parameters are very intuitive and have a direct relationship with the experimental conditions. Moreover, it is not necessary to define other thresholds like the maximum distance between clusters, which is hard to estimate but commonly used in tracking algorithms.

We implement Spikes_Link with Wave_clus as its spike sorter and used long-term simulated recordings with drifting neurons to validate the method, comparing its performance with the standard Wave_clus and Combinato. Spikes_Link obtained significantly better values for all the tested metrics: accuracy, recall, precision, number of additional classes, and Adjusted Rand Index. Currently, we are in the process of using real long-term recordings to fully validate the utility of the proposed method.

It could be possible to use Spikes_Link in conjunction with SpikeInterface (Buccino et al., 2019), a framework that unifies multiple spike sorting methods. First, a comparison of the results in a single block could determine which spike sorting algorithm is the best for a given type of recording. Then, with a proper interface between both frameworks, all the methods available on SpikeInterface will be available as Spikes_Link sorters.

References

Continuing progress of spike sorting in the era of big data. (2019). *Current Opinion in Neurobiology*, *55*, 90–96. https://doi.org/10.1016/j.conb.2019.02.007

Neuronal Activity in the Rodent Dorsal Striatum in Sequential Navigation: Separation of Spatial and Reward Responses on the Multiple T Task. (2004). *Journal of Neurophysiology*, *91*(5), 2259–2272. https://doi.org/10.1152/jn.00687.2003

Recording from defined populations of retinal ganglion cells using a high-density CMOS-integrated microelectrode array with real-time switchable electrode selection. (2012). *Journal of Neuroscience Methods*, *211*(1), 103–113. https://doi.org/10.1016/j.jneumeth.2012.08.017

Long Term Recordings with Immobile Silicon Probes in the Mouse Cortex.. (2016). *PLoS One*, *11*, e0151180.

A novel versatile hybrid infusion-multielectrode recording (HIME) system for acute drug delivery and multisite acquisition of neuronal activity in freely moving mice. (2015). *Frontiers in Neuroscience*, *9*. https://doi.org/10.3389/fnins.2015.00425

Chronic multisite, multielectrode recordings in macaque monkeys. (2003). *Proceedings of the National Academy of Sciences*, *100*(19), 11041–11046. https://doi.org/10.1073/pnas.1934665100

Functional Stability of Dorsolateral Prefrontal Neurons. (2004). *Journal of Neurophysiology*, *92*(2), 1042–1055. https://doi.org/10.1152/jn.00062.2004

High-Density Long-Lasting, and Multi-region Electrophysiological Recordings Using Polymer Electrode Arrays. (2019). *Neuron*, *101*(1), 21–31.e5. https://doi.org/10.1016/j.neuron.2018.11.002

Surgical Training for the Implantation of Neocortical Microelectrode Arrays Using a Formaldehyde-fixed Human Cadaver Model. (2017). *Journal of Visualized Experiments*, *129*. https://doi.org/10.3791/56584

Single-cell recordings in the human medial temporal lobe. (2015). *Journal of Anatomy*, *227*(4), 394–408. https://doi.org/10.1111/joa.12228

Novel electrode technologies for neural recordings. (2019). *Nature Reviews Neuroscience*, *20*(6), 330–345. https://doi.org/10.1038/s41583-019-0140-6

Variance and invariance of neuronal long-term representations. (2017). *Philosophical Transactions of the Royal Society B: Biological Sciences*, *372*(1715), 20160161. https://doi.org/10.1098/rstb.2016.0161

Steady or changing? Long-term monitoring of neuronal population activity. (2013). *Trends in Neurosciences*, *36*(7), 375–384. https://doi.org/10.1016/j.tins.2013.03.008

Recording from the same neurons chronically in motor cortex. (2012). *Journal of Neurophysiology*, *107*(7), 1970–1978. https://doi.org/10.1152/jn.01012.2010

Recording Chronically From the Same Neurons in Awake Behaving Primates. (2007). *Journal of Neurophysiology*, *98*(6), 3780–3790. https://doi.org/10.1152/jn.00260.2007

Tracking neurons recorded from tetrodes across time. (2004). *Journal of Neuroscience Methods*, *135*(1-2), 95–105. https://doi.org/10.1016/j.jneumeth.2003.12.022

Tracking single units in chronic large scale, neural recordings for brain machine interface applications. (2014). *Frontiers in Neuroengineering*, *7*. https://doi.org/10.3389/fneng.2014.00023

Single-Unit Stability Using Chronically Implanted Multielectrode Arrays. (2009). *Journal of Neurophysiology*, *102*(2), 1331–1339. https://doi.org/10.1152/jn.90920.2008

de la Prida, L. M. (Ed.). (2016). Reliable Analysis of Single-Unit Recordings from the Human Brain under Noisy Conditions: Tracking Neurons over Hours. *PLOS ONE*, *11*(12), e0166598. https://doi.org/10.1371/journal.pone.0166598

Spike sorting: Bayesian clustering of non-stationary data. (2006). *Journal of Neuroscience Methods*, *157*(2), 303–316. https://doi.org/10.1016/j.jneumeth.2006.04.023

A Bayesian Clustering Method for Tracking Neural Signals Over Successive Intervals. (2009). *IEEE Transactions on Biomedical Engineering*, *56*(11), 2649–2659. https://doi.org/10.1109/tbme.2009.2027604

A non-parametric Bayesian approach for clustering and tracking non-stationarities of neural spikes. (2014). *Journal of Neuroscience Methods*, *223*, 85–91. https://doi.org/10.1016/j.jneumeth.2013.12.005

Model-based spike sorting with a mixture of drifting t -distributions. (2017). *Journal of Neuroscience Methods*, *288*, 82–98. https://doi.org/10.1016/j.jneumeth.2017.06.017

Automated long-term recording and analysis of neural activity in behaving animals. (2017). *ELife*, *6*. https://doi.org/10.7554/elife.27702

Kalman filter mixture model for spike sorting of non-stationary data. (2011). *Journal of Neuroscience Methods*, *196*(1), 159–169. https://doi.org/10.1016/j.jneumeth.2010.12.002

Improved Spike-Sorting By Modeling Firing Statistics and Burst-Dependent Spike Amplitude Attenuation: A Markov Chain Monte Carlo Approach. (2004). *Journal of Neurophysiology*, *91*(6), 2910–2928. https://doi.org/10.1152/jn.00227.2003

An online spike detection and spike classification algorithm capable of instantaneous resolution of overlapping spikes. (2009). *Journal of Computational Neuroscience*, *29*(1-2), 127–148. https://doi.org/10.1007/s10827-009-0163-5

A novel and fully automatic spike-sorting implementation with variable number of features. (2018). *Journal of Neurophysiology*, *120*(4), 1859–1871. https://doi.org/10.1152/jn.00339.2018

A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. (2018). *ELife*, *7*. https://doi.org/10.7554/elife.34518

Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. (2017). https://doi.org/10.1101/101030

A Fully Automated Approach to Spike Sorting. (2017). *Neuron*, *95*(6), 1381–1394.e6. https://doi.org/10.1016/j.neuron.2017.08.030

Unsupervised Spike Sorting for Large-Scale High-Density Multielectrode Arrays. (2017). *Cell Reports*, *18*(10), 2521–2532. https://doi.org/10.1016/j.celrep.2017.02.038

SMOTE: Synthetic Minority Over-sampling Technique. (2002). *Journal of Artificial Intelligence Research*, *16*, 321–357. https://doi.org/10.1613/jair.953

How many neurons can we see with current spike sorting algorithms?. (2012). *Journal of Neuroscience Methods*, *211*(1), 58–65. https://doi.org/10.1016/j.jneumeth.2012.07.010

Past present and future of spike sorting techniques. (2015). *Brain Research Bulletin*, *119*, 106–117. https://doi.org/10.1016/j.brainresbull.2015.04.007

Properties of the Hubert-Arable Adjusted Rand Index.. (2004). *Psychological Methods*, *9*(3), 386–396. https://doi.org/10.1037/1082-989x.9.3.386

MONIC. (2006). *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 06*. https://doi.org/10.1145/1150402.1150491

Learning from streaming data with concept drift and imbalance: an overview. (2012). *Progress in Artificial Intelligence*, *1*(1), 89–101. https://doi.org/10.1007/s13748-011-0008-0

SpikeInterface a unified framework for spike sorting. (2019). https://doi.org/10.1101/796599