Appendix: Algorithms

The algorithms implemented to create the visual component *storm graph structure* consist of: (1) a cluster-based layout that groups nodes vertically depending on their spatial closeness, and (2) a collapsible feature that simplifies large consecutive trains of storm cell nodes, in the temporal axis (horizontally). These trains do not present splitting or merging branches.

For the clusterization of the storm branches, the algorithm groups nodes based on the DBScan clustering technique (see Algorithm 1). When computing the layout of the graph, the nodes that correspond to a same geographic cluster are grouped together in the storm graph structure. The layout is created considering larger spaces between clusters of nodes, namely storm branches, and making storm branches geographically distant, also farther distant from their sibling clustered storm branches in the graph.

Algorithm 1: Clustering of nodes in each time step

In: timesteps (list of dates), stepsByDate
Result: Steps clusters

1: $clustersByStepId \leftarrow \{\}$
2: $clustersByDate \leftarrow \{\}$
3: **for all** $data \in timesteps$ **do**
4: $steps \leftarrow stepsByDate[date]$
5: $clusters \leftarrow getDBScanClusters(pointsForStep)$
6: $clustersList$.push($clusters$)
7: **end for**
8: return $clustersList$

The collapsible feature (see Algorithm 2), starts with no potential interval (indicated by *intervalEnd* and *IntervalStart* set to -1). The first iteration of the loop checks if the last time step could be the end of a new potential interval. For this case, nodes should not be the result of a join (Line 8). The next iteration will try to grow the interval by setting a new start for the interval, or by making it grow if *intervalStart* is not -1. The condition for adding the time step is that its nodes do not result from a join or generate a split (conditions in Line 5 and Line 12). If these conditions are not met, then the interval can be closed if it contains more than one time step. At this point, no potential interval exists and so *intervalEnd* and *intervalStart* are set to -1, so a new end can be searched in the next iteration.

Algorithm 2: Assembing collapsible node chains in the graph

In: dates (list of dates), nodes by date
Result: An array of intervals that can be collapsed

1: $intervalEnd \leftarrow$ -1
2: $intervalStart \leftarrow$ -1
3: $intervals \leftarrow$ []
4: **for** di = dates.length - 1 TO 0 **do**
5: $eys \leftarrow$ do all nodes in time step di have less that one parent and one child?
6: **if** $intervalEnd = -1$ **then**
7: $ps \leftarrow$ do all nodes in time step $di+1$ have less than one parent?
8: **if** eys and ps **then**
9: $intervalEnd \leftarrow di$
10: **end if**
11: **else**
12: $is \leftarrow$ do all nodes in time step $d-1$ have less than one child?
13: **if** eys and is **then**
14: $intervalStart \leftarrow di$
15: **else**
16: **if** $intervalStart < intervalEnd$ and $intervalStart \neq -1$ **then**
17: $newInterval \leftarrow \{intervalStart, intervalEnd\}$
$intervals$.push($newInterval$)
18: **end if**
19: $intervalEnd \leftarrow -1$
20: $intervalStart \leftarrow -1$
21: **end if**
22: **end if**
23: **end for**
24: **if** $intervalEnd \neq -1$ and $intervalStart \neq intervalEnd$ **then**
25: $newInterval \leftarrow \{0, intervalEnd\}$
$intervals$.push($newInterval$)
26: **end if**
27: return $intervals$