Detection, Tracking, and Visualization of Spatial Event Clusters for Real Time Monitoring

Natalia Andrienko[1,2], Gennady Andrienko[1,2], Georg Fuchs[1], Salvatore Rinzivillo[3], Hans-Dieter Betz[4]
[1] Fraunhofer Institute IAIS, Sankt Augustin, Germany; [2] City University London, UK
e-mail: 1st name.2nd name@iais.fraunhofer.de
[3] Institute of Information Science and Technology (ISTI), National Research Council (CNR), Pisa, Italy
e-mail: rinzivillo@isti.cnr.it
[4] nowcast GmbH, Munich, Germany
e-mail: hdbetz@nowcast.de

***Abstract*—Spatial events, such as lightning strikes or drops in moving vehicle speed, can be conceptualized as points in the space-time continuum. We consider real time monitoring scenarios in which the observer needs to detect significant (i.e., sufficiently big) spatio-temporal clusters of events as soon as they occur and track the further evolution of these clusters. Isolated spatial events and small clusters are of no interest (i.e., treated as noise) and should be hidden from the observer to avoid attention distraction and perceptual overload. The existing methods for stream clustering cannot enable on-the-fly separation of event clusters from the noise and immediate presentation of significant clusters and their evolution. We propose a novel algorithm tailored to this specific task and a visual analytics system that supports event stream monitoring by presenting detected event clusters and their evolution to the observer in real time.**

Keywords—spatio-temporal data, data stream clustering, visual analytics

I. INTRODUCTION

Spatial events are physical or abstract entities, such as lightning strikes or mobile phone calls, which occur at some time moments at particular locations in space and have limited existence times. A spatial event is characterized by its start and end times (which may coincide), spatial coordinates, and, possibly, some thematic attributes. We assume that occurrences of spatial events are registered, e.g., by sensors, and corresponding data records are immediately sent to a server. The resulting data stream needs to be monitored.

We consider monitoring scenarios in which any individual spatial event is of no interest and only spatio-temporal event clusters (i.e., occurrences of multiple events closely in space and time) may require observer's attention. For example, moving vehicles may emit low speed events when their speed drops below a certain threshold. It is neither feasible nor meaningful to attend to every such event, but a spatio-temporal cluster of low speed events sent by several cars may deserve observer's attention as a possible indication of a traffic jam. After detecting a cluster, the observer may need to trace its further evolution, i.e., changes in the number of events, number of vehicles involved, spatial location, shape, and extent. Our goal is to support the observer in detecting the emergence and tracking the evolution of spatial event clusters in real time.

The research has been partly supported by EU in projects INSIGHT "Intelligent Synthesis and Real-time Response using Massive Streaming of Heterogeneous Data" (http://insight-ict.eu/) and SoBigData Research Infrastructure.

The task of separating spatio-temporal event concentrations (clusters) from scattered events (noise) requires an analog of a density-based clustering method capable to process a data stream and discover clusters on the fly. However, the existing stream clustering methods are oriented to somewhat different problem settings. We have developed a new algorithm that fulfills this specific task. The algorithm is integrated in a visual analytics environment that supports the monitoring by visually presenting the evolving situation to the observer.

Our research contribution consists of the following components. First, we propose a novel algorithm specially designed for the task of real-time detection and tracking of spatio-temporal clusters of spatial events (section III). Second, we propose an interactive visual interface integrating the event clustering algorithm and supporting real-time event stream monitoring (section IV). Third, we describe several case studies demonstrating the application of the algorithm to data with different properties and its capability to track event clusters with different behaviors (section V). We also theoretically and empirically compare our algorithm to the classical density-based clustering (sections III.G and VI).

II. RELATED WORKS

A. Stream clustering

1) Existing approaches

A number of methods for data stream clustering have been proposed in the data mining literature; however, they are not suitable for our specific needs due to focusing on a different problem. In fact, the main problem they address is the memory limitation. As individual data records are too numerous to be stored, the main goal of the streaming algorithms is effective summarization and compact representation of an incoming data stream. The methods summarize incoming data on the fly and keep only summaries (*micro-clusters*) but not the original data items. Many streaming algorithms take a two-phase approach: micro-clusters are created and maintained during an online phase and post-processed (e.g., merged into larger clusters) during an offline phase. This general framework is instantiated with different approaches to creating micro-clusters. The main representatives are CluStream [1] and DenStream [2] doing partition-based and density-based clustering, respectively.

CluStream partitions an initial portion of a stream into k micro-clusters. When a new data point d appears, it tries to fit d into one of the current micro-clusters, while satisfying the constraints on the maximum number of clusters k and maximum boundary R. DenStream identifies micro-clusters with a maximal radius *Eps* based on the concepts of core object and density adopted in density-based clustering. Unlike in CluStream, the number of micro-clusters is not bounded. Both approaches rely on a following offline phase, in which micro-clusters are merged into macro-clusters.

2) Differences of our problem settings

Our problem setting and requirements differ from those of the existing methods in several respects. First, emerging significant clusters need to be detected in real time and immediately shown to the observer, permitting no reliance on off-line post-processing. Second, clusters may emerge, evolve (expand, shrink, move, change shape, split, merge), and disappear, excluding the approaches assuming a constant number of clusters, like [1]. Third, optimizing the use of the main memory is not a primary goal. While it is not supposed to keep all incoming data records in the main memory (which is neither feasible nor needed), it can be assumed that the available memory is sufficient for keeping all micro-clusters that may co-exist within a time interval $[t_c-\Delta T, t_c]$, where t_c is the current time moment and ΔT is a user-chosen time window. Fourth, old micro-clusters (where the latest event is older than $t_c-\Delta T$) are not of interest anymore and may be discarded.

According to these settings, we propose a hybrid approach in which micro-clusters are built and updated similarly to [1], but without limiting the maximal number. They are merged online into larger clusters of arbitrary sizes and shapes by exploiting *k-connectivity*, similarly to [2].

3) Other works on clustering

Our proposed algorithm produces results similar to those of standard density-based clustering methods, such as DBSCAN [3] and OPTICS [4]. These methods as such are not meant for stream clustering. To detect clusters of moving objects in a set of trajectories (but not in a data stream), Kalnis et al. [5] applied DBSCAN to data subsets from different time steps. To establish a correspondence between clusters from two runs, the clusters were checked for containing a sufficient number of common objects. This approach is suitable for trajectories but not for arbitrary events (clusters obtained for different time steps would not include any common objects). For events, Peca et al. [6] propose to apply DBSCAN to overlapping time intervals and join clusters from consecutive runs when they have common members. This approach involves repeated re-clustering of the same events, which is rather costly, and so is the operation of establishing a correspondence between clusters based on their members. Besides, both works apply clustering to discrete time steps. We do clustering in continuous time, which is more suitable for streaming settings.

B. Visual analytics of event streams

The task of real-time analysis and monitoring of streams of generic spatial events has not yet been addressed in visual analytics. A popular research topic is analysis of text streams, such as streams of tweets, with a focus on text analysis, e.g., for detecting unusual bursts of some terms. Visually supported real-time detection of spatial clusters of thematically related georeferenced tweets is presented in [7]. Relevant terms (e.g., terms indicating anomalies) are extracted from the texts. Each term occurrence is treated as a separate spatial event. A variant of the Lloyd's clustering algorithm is applied to occurrences of the same term. Initially, a single cluster for each term is created. When the average squared distance of the members to the cluster center exceeds a chosen threshold, the cluster is split into two new clusters. Significant clusters (i.e., sufficiently big while compact) are presented to the user in the form of term clouds on a map display. Neither the spatial extents of the clusters nor their evolution are shown to the user.

Our method and system have been designed for generic spatial events. Section V.A shows that they can be applied, in particular, to georeferenced tweets, assuming that relevant tweets (e.g., containing terms of interest) are selected from the overall stream by appropriate filtering tools. By involving the extension described in section III.F, our method can also be used to find clusters of occurrences of the same term.

The need of tracking and presenting cluster evolution requires a different method for event clustering than the one used in [7]. The latter is not suitable for tracking spatially expanding clusters: it initially assumes that there is a single cluster and later can only subdivide this cluster. It also cannot detect dense clusters of arbitrary shapes but constructs only "round" clusters, similarly to k-means. This approach cannot adequately identify spatially elongated clusters, such as a cluster of low speed events located along a street (section V.B).

III. Our Approach to Event Stream Clustering

A. Definitions

Definition 1. A **spatial event** is a point in the space-time continuum defined by a triple *(x, y, t)*, where x and y are the coordinates (such as longitude and latitude) of the event in the space and t is the time moment or interval when the event occurred.

Note 1. According to our use cases, we consider two-dimensional space. The algorithmic part of the approach is straightforwardly extendable to three-dimensional space, but not the visualization part involving 2D maps and 3D space-time cubes, where the third dimension represents time.

Definition 2. An **active event *e*** is a spatial event such that its life time t has a non-empty intersection with the interval $[t_c-\Delta T, t_c]$, where t_c is the current moment and ΔT is a user-specified maximal temporal gap.

Definition 3. An **event circle *C*** is a group of active events that fits in a circle with a user-specified maximal radius R. Formally, $\boldsymbol{C} = \{e \mid d(e, c) \le \boldsymbol{R}\}$, where $d(p, q)$ is the distance of two points p and q in space and c is the center of the circle, defined as $c = \underset{\langle x,y \rangle}{\operatorname{argmin}} \sum_{e \in C} d(e,(x,y))$.

Note 2. The term "event circle" denotes a group of events, not a geometric figure. This meaning roughly corresponds to what is meant by "micro-cluster" in [2]. We use the term

'event circle' as more specific and expressive. In particular, an event circle may consist of a single event.

Note 3. Assuming that each event belongs to an event circle whose center is the nearest to this event, the minimal distance between two event circles may not be less than R.

Definition 4. Two event circles C_1 and C_2 are **connected** if there exists at least one active event e, called **connecting event**, such that $d(e, c_1) \leqslant R$ and $d(e,c_2) \leqslant R$, where c_1 and c_2 are the centers of the two event circles, respectively.

Note 4. The maximal possible distance between the centers of two connected circles is not more than $2*R$. For any circle C, there may be at most 12 connected circles, which may happen in a case when the centers of all connected circles lie in the maximal distance $2*R$ from the center of C and in the minimal distance R (see Note 3) from each other.

Note 5. One event may be a connecting event for at most 7 circles, which may happen in a case when the event lies in the center of one circle C and the centers of all others lie exactly in the minimal possible distance R (see Note 3) from the center of C and from each other.

Definition 5. Two event circles C_1 and C_2 are **directly *k*-connected** (denoted with $C_1 \equiv C_2$) if there exist at least k (a user-specified number) connecting events for C_1 and C_2.

Note 6. We emphasize the fact that the property of connectedness is dynamic, i.e., event circles may become connected at some time but later disconnect.

Note 7. Our definition of connectedness is similar to that in algorithm AING [8], where analogs of micro-clusters are represented as graph nodes. When a connecting data point appears, an edge is created between the respective nodes.

Note 8. If $C_1 \equiv C_2$ than the following conditions hold: $C_2 \equiv C_1$ (symmetry), $C_1 \equiv C_1$ and $C_2 \equiv C_2$ (identity).

Definition 6. Two event circles C_1 and C_2 are ***k*-connected** (denoted with $C_1 \approx C_2$) if they are directly k-connected or there exists a sequence of event circles $[C_i, C_{i+1}, \ldots, C_m]$ such that $C_1 \equiv C_i$, $C_i \equiv C_{i+1}$, …, $C_m \equiv C_2$.

Definition 7. A **union** of event circles $\boldsymbol{U}$ is a set of event circles $\{C_i\}$ such that for each $C_j \in U$, it holds $C_j \approx C_k$, for all $C_k \in U$.

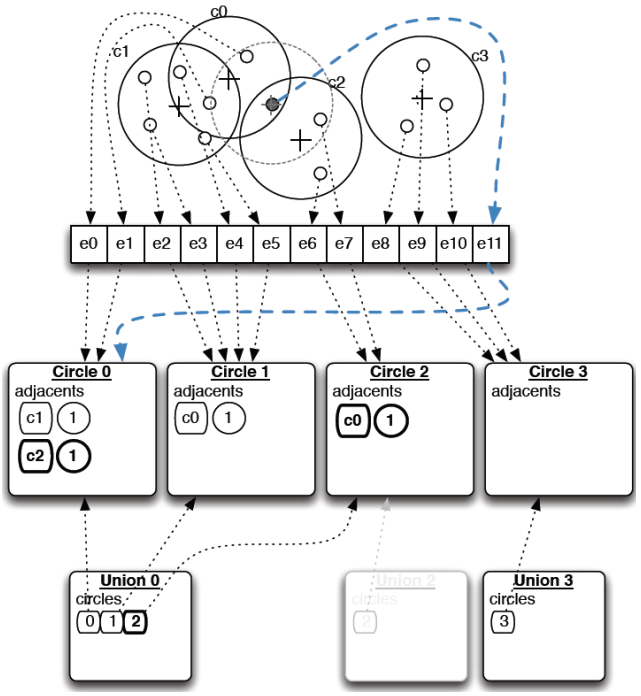
Note 9. A union may, in particular, consist of a single event circle.

Definition 8. The **center** of a union is the center of the smallest geometric circle enclosing all its events.

Definition 9. A **significant event cluster** is a union that includes at least $\boldsymbol{N}$ (a user-specified number) spatial events. The value N is called **significance threshold.**

Note 10. As soon as a significant event cluster emerges, it must be presented to the observer. The further evolution (i.e., all changes) of this event cluster must be tracked and also presented to the observer until the cluster vanishes.

Fig. 1. A schematic illustration of the data structures and the changes made when a new event arrives. The changes are marked with thicker lines and texts in bold. The new event e11 is assigned to the circle with closest center: c0. Since e11 is within distance R from the center of the circle c2, it is a connecting event for c0 and c2. The adjacency lists of c0 and c2 are updated to maintain the counts of connecting events. If a count becomes equal to k (here we set k=1), the two circles are joined in the same union. The algorithm select the union U0 as the destination and puts the two circles in this container. The other union, U2, is discarded.



B. *Data structures*

Events are maintained in a chronologically ordered list EL. The algorithm keeps in the memory only active events (Definition 2) and removes older events. Each event $e \in EL$ has a non-empty list of references to circles $\{c_j\}$ such that $d(e,c_j.\text{center}) \leqslant R$. The first circle in the list is the circle c^* containing e, and the remaining circles, if any, are the circles connected with c^* by e. The list of circles is denoted $e.circles$.

Each circle c_i has a reference to its union, denoted $c_i.union$, and an adjacency list $c_i.adjacents$ containing references to circles connected with c_i by one or more events. For each circle c_j in this list, there is a count of the connecting events for c_i and c_j. When a new connecting event for c_i and c_j is added to EL, the count is increased by one. When a connecting event for c_i and c_j is removed from EL, the count is decreased.

The data structures and the changes made when a new event arrives are schematically represented in Fig. 1.

C. *Algorithm*

The algorithm consists of repeated execution of the main loop triggered by either one or more new event records appearing in the data stream or a tick event sent by the clock. The frequency of sending tick events is chosen by the user.

Input: current time t_c; set of new event records E (E may be empty in the case when the loop is triggered by a tick event).

Parameters: maximal event circle radius R; maximal time gap ΔT; minimal number of connecting events for uniting event circles k; minimal significant cluster size N.

Main loop:

1. Discard events older than t_c-ΔT and remove empty circles.
2. Assign each new event from set E to a circle.
3. Unite k-connected circles.
4. Separate unions after circle removal.

These steps of the main loop are described in detail below.

1) Discard old events

1. while (head(EL).time < t_c-ΔT):
   - 1.1. h <- pop(EL); C <- h.circles; c^* <- pop(C):
   - 1.2. c^* <- $c^* \setminus \{h\}$
   - 1.3. if c^* is empty:
     - 1.3.1. for all circles $c_i \in C$:
       - 1.3.1.1. remove c^* from c_i.adjacents
     - 1.3.2. Let UC=c^*.union.circles; UC <- $UC \setminus \{c^*\}$
     - 1.3.3. discard c^*
   - 1.4. else, for all circles $c_i \in C$:
     - 1.4.1. c^*.adjacents(c_i).count -=1
     - 1.4.2. c_i.adjacents(c^*).count -=1

The search and removal of a single outdated event is performed in constant time. When a circle c^* is discarded, the updates of the adjacent connected circles are executed in time $O(|h.\text{circles}|\text{-}1)$; otherwise, the counts of connecting events are updated also in time $O(|h.\text{circles}|\text{-}1)$. Since $|h.\text{circles}| \leqslant 7$ (see note 5), the time may be considered constant.

2) Assign each new event to a circle

Each new event e is associated with an event circle. If there exists a circle c_i such that $d(e, c_i.center) \leqslant R$, it is added to c_i, and the position of the center is updated. If not, a new event circle consisting of only this event is created.

For an effective search of candidate event circles for including new events, we use a spatial index as proposed in [9]. It is based on a spatial grid G covering the observed territory, with square cells having the widths and heights equal to R. Each cell has a list of references to event circles the centers of which are located in the cell. By construction, the list may include at most 4 circles, if the positions of their centers are close to the corners of the cell (see Note 3). To find candidate circles for a new event, it is sufficient to check the grid cell $G_{i,j}$ containing this event and its neighboring cells from $G_{i-1,j-1}$ to $G_{i+1,j+1}$, i.e., at most 9 cells (3×3). The cell $G_{i,j}$ is determined from event coordinates by a simple computation taking constant time. The number of circles that need to be checked is at most 16. This upper bound corresponds to a hypothetical case when all circle centers are located at the corners of the cells (a 3×3 grid has only 16 cell corners rather than 36 because neighboring cells have common corners). Hence, the search of candidate circles for including each new event e can be done in constant time $O(16)$. The upper bound for the number of candidate circles that may exist for e is 7 (see Note 5).

Let $CC=\{c_{i1},c_{i2},\ldots,c_{im}\}$ be the set of candidates and c^* the closest circle to e, i.e. $d(e, c^*.center) \leqslant d(e, c_k.center)$ for any $c_k \in CC$. If CC is empty, a new circle c^* is created containing the new event, and CC is set to $\{c^*\}$. Otherwise, e is put in c^*, and c^* is moved to the head of CC. The list CC is attached to the event e, i.e., $e.circles = CC$. For each pair of circles $c_i \in CC$ and $c_j \in CC$, the adjacency lists of c_i and c_j are updated to include each other along with the actual counts of connecting events. This operation requires quadratic time with respect to the length of the list CC; however, given the limited length of CC (at most 7), the time is $O(21)$, where 21 = 7*6/2.

2. For each new event e,
   - 2.1. $CC=\{c_{i1},c_{i2},\ldots,c_{im}\}$ such that $d(e,c_{ij}.center) \leq R$ for any j
   - 2.2. Select c^* such that $d(e,c^*.center) <= d(e,c_j.center)$, for any c_j in CC
   - 2.3. If CC is empty, create a new event circle c^* = {}; CC={ c^*}; c^*.union = u^*
   - 2.4. Else, move c^* to the head of CC: CC = {c^*, …}
   - 2.5. Add e to c^*; e.circles <- CC
   - 2.6. For each $c_i \in CC$ and $c_j \in CC$, $i \neq j$,
     - 2.6.1. c_i.adjacents(c_j).count +=1
     - 2.6.2. c_j.adjacents(c_i).count +=1

3) Unite k-connected circles

The algorithm checks if any of the candidate event circles have just become k-connected, i.e., after steps 2.6.1 and 2.6.2, the counts exactly equal k. If so, the algorithm joins the k-connected event circles by merging two or more unions.

3. For each event circle c_i,
   - 3.1. Let UC = ∅
   - 3.2. For each event circle c_j in c_i.adjacents,
     - 3.2.1. If c_i.adjacents(c_j).count = k and c_i.union ≠ c_j.union,
       - 3.2.1.1. CU = CU ∪ {c_j }
   - 3.3. Let U_1=c_i.union
   - 3.4. For each event circle c_j in CU
     - 3.4.1. Let U_2=c_j.union
     - 3.4.2. Let U*,U^- = select(U_1, U_2)
     - 3.4.3. U*.circles = U*.circles ∪ U^-.circles
     - 3.4.4. Remove(U^-)
     - 3.4.5. U_1= U*
   - 3.5. For each event circle c_j in U_1.circles,
     - 3.5.1. c_j.union <- U_1

Merging two unions can be computed in time $O(|c_i.\text{adjacents}|)$. Since $|c_i.\text{adjacents}| \leqslant 12$ (see Note 4), the time is $O(12)$. The candidate unions for merging are determined directly from pointers within c_i.adjacents. To maintain the cluster identities, we introduce different strategies in line 3.4.2, where the destination union is selected and the other is removed. The strategies are discussed in section III.D.

4) Separate unions after circle removal

4. For each union U that lost any circle in step 1.3.2:
   - 4.1. While |U.circles| > 1, do the following:
     - 4.1.1. Take any circle $c_i \in U$; U.circles <- U.circles\{ c_i }
     - 4.1.2. Create U': U'.circles = { c_i }
     - 4.1.3. For each $c_m \in U'$,
       - 4.1.3.1. Find $CC \subseteq c_m$.adjacents where c_m.adjacents(c_j).count≥k for any $c_j \in CC$
       - 4.1.3.2. U.circles <- U.circles\CC
       - 4.1.3.3. U'.circles <- U'.circles ∪ CC
   - 4.2. If U.circles = ∅, remove U.

The procedure picks an arbitrary circle from U and follows the circle's links to select its k-connected circles. The group of k-connected circles is removed from U and forms a new union U'. This is done until U gets empty or a single circle remains. In the worst case, when each initially chosen circle is located at the periphery of U, the procedure takes time $O(|U.\text{circles}|)$. The cost of step 4.1.3.1 may be considered constant since $|c_m.\text{adjacents}| \leqslant 12$ for any circle c_m.

D. Maintaining cluster identities

A significant event cluster is a union containing at least N events (Definition 9). Significant clusters are presented to the observer as soon as they emerge. Since than, all changes occurring to them are tracked and also presented to the analyst. The possible types of change are: adding new events, removing old events, merging with other unions, and splitting into smaller unions. Adding and removing events do not pose any problems concerning the cluster identities; however, merging and splitting of unions require special care, as described below.

When two or more unions are merged into a single union, presenting the result as a new cluster that has just emerged may confuse the observer. Merges may occur very frequently, and presenting their results as new clusters would make the changes too difficult to track. It is more appropriate to treat a single union U resulting from merging several unions U_1, …, U_m as the continuation of one of these unions.

Different criteria can be used for choosing one union U* among U_1, …, U_m to be treated as absorbing the others. It may be the largest one, or the oldest one, or the most central one, i.e., such that the center of U* is the closest to the center of the resulting union U. We use the latter criterion. The rationale is that the current positions of the cluster centers and their recent changes are visually presented to the user. By taking the most central union among U_1, …, U_m as the previous state of the resulting union U, we minimize the abruptness of the change in the position of the cluster center, which makes it easier for the observer to keep track of the changes.

When one union U is split into two or more unions U_1, …, U_m, a similar problem arises: which of the unions U_1, …, U_m should be treated as the continuation of the union U? Here, the same choice criterion is used as in the case of merging. The remaining unions are treated as new clusters.

E. Tracking cluster evolution

The evolution of each union is tracked by maintaining a sequence of state records. The first state record is generated when a union is created. A new state record is generated and added to the sequence when any change occurs. The last state record is created when the union dies due to ageing or absorption. A state record includes the following data: the time of this state, the current number of member events, the total number of the member events since the emergence of the union (including also the old events that have been removed), the current position of the center, the radius of the circumcircle, and the spatial convex hull of the current member events.

F. Extensions of the base algorithm

The algorithm can additionally account for thematic attributes of the events. For example, low speed events from moving vehicles may have an attribute 'movement direction'. The algorithm can ensure that only events with similar directions are put together in the same union (the user needs to specify the maximal allowed difference in the directions). Generally, the user can choose a set of thematic attributes A_1, …, A_n and specify a tuple of corresponding difference thresholds t_1, …, t_n. If A_i is a nonnumeric attribute, the corresponding threshold $t_i = 0$. An event circle c is considered as a candidate for including an event e if, first, the condition of the step 2.1 of the main loop holds and, second, the differences between the values of the attributes A_1, …, A_n for e and the average values for c do not exceed the respective thresholds t_1, …, t_n. The set CC in the step 2.1 of the main loop is composed of the event circles satisfying both these conditions.

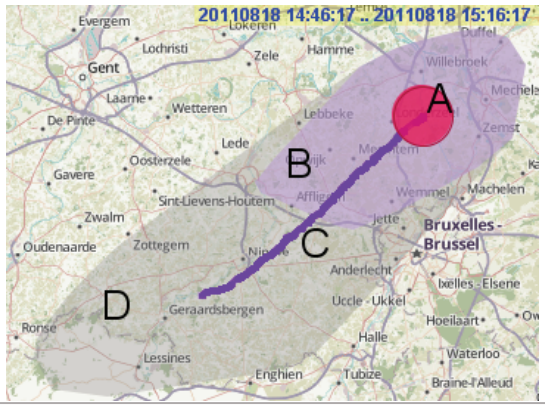
Another extension is accounting for the event sources, such as vehicles sending low speed events or Twitter users. For each union, the algorithm can maintain a list of distinct event sources. Unions where all events come from too few distinct sources (i.e., fewer than a user-specified number D) are treated as insignificant even when their sizes reach the threshold N.

G. Comparison to density-based clustering

In density-based clustering (DBC), clusters are defined as areas of higher density than in the remainder of the dataset. The density is defined as the number of data points within a given distance ε from a chosen point. While our definition of clusters differs from this, some similarities exist. Assuming that our parameter R corresponds to ε, the event density for an event circle c is the sum of the number of its member events and the number of connecting events belonging to adjacent event circles. Event circles can be joined when they have at least k connecting events, which means that the minimal density for each of them needs to be at least k+1 (k connecting events from another event circle plus at least one own member event). A union consisting of a single event circle is treated as a significant cluster if it contains at least N events. In case of taking $N < k$+1, some of the resulting clusters may have lower density than k+1. Such a choice is not meaningful because it leads to an inconsistency between the density requirements for isolated event circles and for unions of two and more event circles. Therefore, we assume that $N \geq k$+1.

Hence, like DBC, our algorithm finds areas of sufficiently high density, that is, at least k+1 events within a radius R. However, there are differences in how objects are attached to clusters. DBC would attach an event to an already existing cluster when there is a sufficient number (k) of other events in in a circle of radius R around the event (R = ε). Our algorithm attaches an event to an existing cluster (union) when the event lies within the distance R from the center of one of the event circles in this union. This condition may be more constraining: the event may be within distance R to k or more member events of the circle but still far from the circle center. In this case, our algorithm will create a new event circle consisting of the new event. This new circle may be later connected to the existing union if at least k connecting events come within ΔT, but the algorithm does not check if any of the past events connect the new circle with already existing circles (our algorithm attends to each event only once). This is different from the behavior of the classical (not incremental) DBC.

Fig. 3. Presentation of a significant cluster on a map. A: the latest position of the cluster center; the circle size is proportional to the number of member events. B: the spatial convex hull of the latest state of the cluster. C: the trajectory of the cluster center made during the time interval $[t_c\text{-}TH, t_c]$. D: the spatial convex hull of all cluster states attained during $[t_c\text{-}TH, t_c]$.



There may also be an opposite case: our algorithm may put a new event in an existing circle when the event has less than k neighbors, whereas DBC would attach such an event to an existing cluster only when it is reachable from some core member of the cluster. Due to these differences, it cannot be expected that the two approaches produce identical results for the same data. Still, very similar results can be expected since the cases mentioned above mostly refer to borderline events rather than really dense areas. A good correspondence between the results of our method and DBC has been confirmed empirically (section VI).

IV. Interactive Visual Interface for Event Stream Monitoring

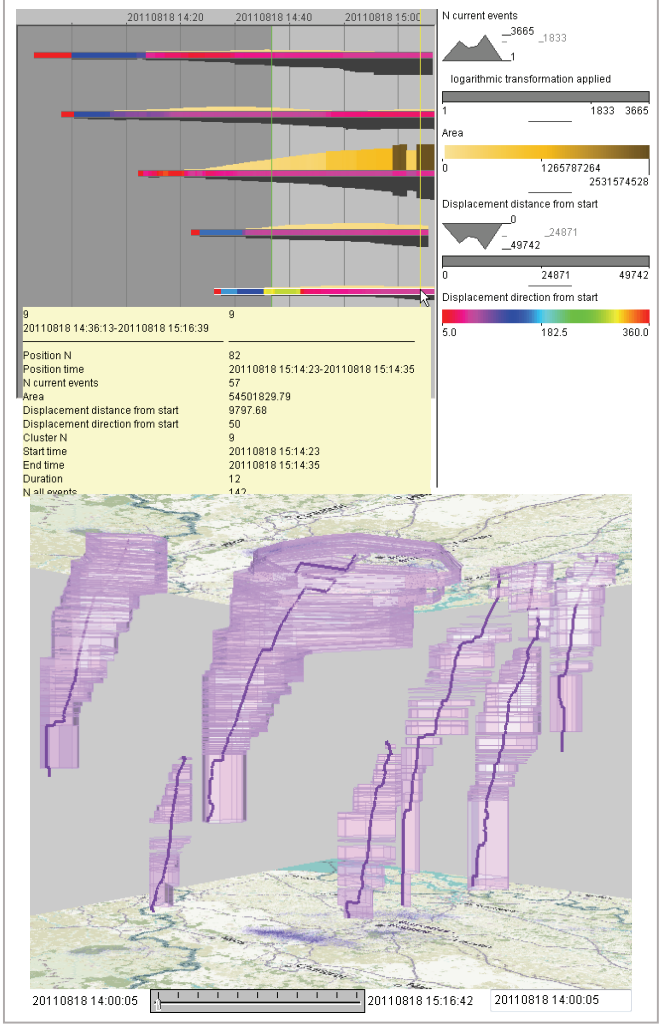
A. Dynamic map

The main visual display supporting the process of event stream monitoring by a human observer is a dynamic map. To reduce the observer's workload, only significant event clusters are presented. When the existing significant clusters change or new significant clusters emerge, the map display is updated. The map shows the situation at the current time moment t_c and the recent history within the time interval $[t_c\text{-}TH, t_c]$, where TH is a user-chosen time horizon. Normally, $TH \geq \Delta T$.

For each significant cluster, the map shows its latest state and the trajectory made by its center during the interval $[t_c\text{-}TH, t_c]$, see Fig. 2. For the latest state, the map shows the spatial convex hull of the cluster, the position of its center, and the cluster size, i.e., the total number of member events since the cluster emergence, which is represented by a proportional size of a circle symbol drawn at the position of the cluster center. The trajectory of the cluster is represented by a line linking the chronologically consecutive positions of the center during the interval $[t_c\text{-}TH, t_c]$. Additionally, a convex hull enclosing all cluster states attained during this time interval is shown. Two convex hulls are visually differentiated, as illustrated in Fig. 2.

As clusters grow, the visual encoding of the cluster sizes is modified, so that the maximal circle size corresponds to the maximal cluster size reached so far. The map legend is updated accordingly. We admit that the modification of the visual encoding may be problematic for the observer, who may fail to notice that cluster sizes increase. We are still looking for a good solution to this problem, such as making the changes in the legend more prominent for attracting observer's attention.

Fig. 2. Additional views allow exploration of the cluster histories. Top: timeline view, with a legend on the right; bottom: space-time cube.



B. Supplementary displays

During the process of monitoring, the time and attention of the observer are typically limited resources, which should not be split between several displays. In our design, no additional displays are required for monitoring. Still, two supplementary interactive displays may be optionally used by the observer or another analyst for exploring the history of the clusters. A timeline view (Fig. 3, top) shows the lifetimes of the clusters and changes of their characteristics throughout the lifetimes. The horizontal dimension represents time. The right part, with a lighter background, corresponds to the interval $[t_c\text{-}TH, t_c]$, and the left part shows the previous history. Significant clusters are represented by horizontal bars, which may be colored or shaded according to values of a selected attribute of the clusters. In Fig. 3, the displacement directions of the clusters are color-coded using a color scale where red corresponds to the north and cyan to the south. To show more attributes,

polygons can be attached above and/or below the bars. The values can be represented by the polygon heights and/or coloring or shading. In our example, the upper polygon heights show the cluster sizes (event numbers) at different time moments, the shading of the upper polygons from light yellow to brown shows the areas covered by the clusters, and the heights of the lower polygons represent the displacement distances of the clusters with respect to their start positions.

In a space-time cube (Fig. 3, bottom), the horizontal plane represents the observed territory and the vertical dimension represents time, the time axis being directed from bottom to top. The clusters are represented by their spatio-temporal convex hulls (in light lilac) and by trajectory lines (in purple) showing the movement of the cluster centers. The cube view can be rotated, tilted, shifted in different directions, and zoomed. The analyst can apply interactive filtering (spatial, temporal, or attribute-based) to focus on the evolution of particular clusters of interest. The filtering affects both the space-time cube and the timeline view.

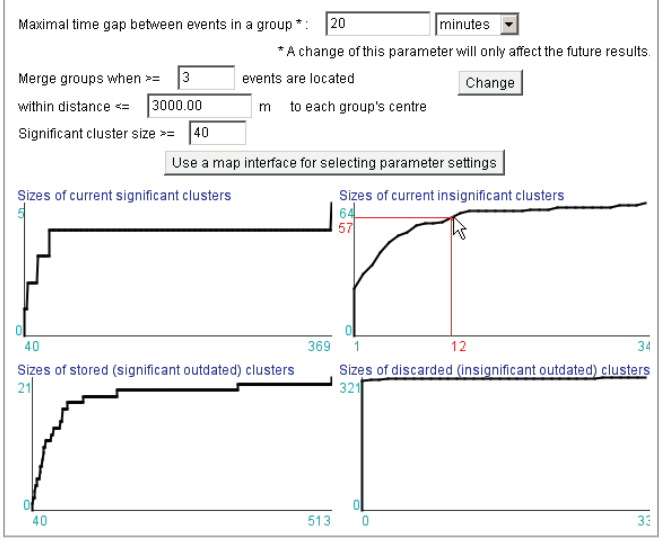
C. *Interactive modification of parameters*

It may not be obvious to the user right at the beginning of the observation what values of the clustering parameters should be chosen. We have created a set of interactive tools allowing the user to change the parameter settings at any time during the monitoring process and helping the user to choose suitable values. When the user changes the settings, the clustering tool takes the new values and uses them from now on. The changes do not affect the previous results, i.e., old clusters, which have been removed from the memory, and the previous histories of the currently existing clusters. The current set of clusters may be affected. Changing the significance threshold N may change the set of clusters shown to the user. Changing the connectivity threshold k may result in some clusters being merged or split. A decrease of the time gap threshold ΔT may lead to some clusters getting old and being removed from the memory. Changing the maximal radius R may potentially affect the connectedness of event circles: when R decreases, some connecting events may lose their status, and when it increases, some events may acquire this status. However, the statuses of the existing events are not re-checked for the efficiency sake.

There are three levels of support to the user in choosing suitable parameter values. On the first level, the user can change the values in real time by looking at four cumulative frequency curves showing the following statistics (Fig. 4): sizes of current significant clusters, sizes of current insignificant clusters, sizes of old significant clusters (such clusters are stored in a database), and sizes of old insignificant clusters that have been discarded. This interface is especially helpful for finding a suitable significance threshold N. Seeing that there are some insignificant clusters with the sizes close to the current N, the user may decrease N for making these clusters visible. Seeing that there are many significant clusters with sizes much larger than N, the user may decide to increase N. If the density of the input event stream varies over time, it may be reasonable to amend N multiple times.

On the second level of support, the user is provided, upon request, with an interactive interface including a map view and a set of controls analogous to those in Fig. 4. The map includes

Fig. 4. Interactive tools for changing clustering parameters.



the active events and clusters that existed in the memory at the moment of creation, i.e., a snapshot of the current situation. The event clustering process continues working, and the main map continues being updated to show the current situation, but the snapshot map does not change when new events come. The events shown in it are "frozen", i.e., not subject to aging and removal from the memory. The map shows both significant and insignificant clusters, distinguishing them by coloring.

For the snapshot, a new instance of the clustering tool is created. The user may interactively change the parameter values, let the tool re-cluster the frozen events using the new settings, and observe the result on the map. The change does not affect the main clustering process. The user can experiment until obtaining a satisfactory result for the frozen event subset and then pass the new parameter settings to the main clustering tool for being applied to the event stream.

On the third level of support, the user can interactively select one or more clusters on the map, choose whether they need to be refined (by splitting) or merged, and let the system find such combinations of values of the parameters k, R, and ΔT which will lead to the desired result. Cluster splitting can be achieved by increasing the value of k and/or decreasing the values of R and ΔT, and cluster merging can be achieved by opposite changes. The parameter N is not modified as it only affects the presentation of the results but not the clustering.

The system systematically tests diverse value combinations using a kind of binary search. For one parameter P, it first doubles or halves the current value p_0, depending on whether it needs to be increased or decreased. If this does not bring the desired effect, it doubles or halves the value again. When the effect is achieved, the system tries to bring the successful value p^* closer to p_0 by repeatedly taking the average between p^* and the last unsuccessful value p^- and assigning it to p^* in case of success and to p^- otherwise, until the difference between p^* and p^- equals 1 in case of k or becomes less than 1% of p^*. For checking the result of each change, the system re-runs the instance of the clustering tool associated with the snapshot. As the tool is applied to the events that are already in the memory,

and the visualization is not involved, it usually takes just a few milliseconds to obtain new results, which permits the use of the procedure in interactive settings.

Using this procedure, the system first finds a successful value k^* for parameter k, keeping the original values R_0 and ΔT_0 of parameters R and ΔT. The combination (k^*, R_0, ΔT_0) is put in the list of successful combinations, which is initially empty. Then, for each value k_i from k_0 to k^*, the system finds a corresponding successful value R^*_i of R, keeping the original value ΔT_0 of ΔT. Before putting each combination (k_i, R^*_i, ΔT_0) in the list of successful combinations, the system checks whether it already contains a combination (k_j, R^*_j, ΔT_0) such that $R^*_i = R^*_j$. In this case, the combination where the value of k is closer to k_0 is kept in the list, and the other one is discarded. The same process as for R is then repeated for ΔT. Finally, for each value k_i from k_0 to k^*, the system takes the corresponding successful values R^*_i and ΔT^*_i and alternately modifies them using the binary search procedure, aiming at bringing them closer to the original R_0 and ΔT_0. Each successful combination that has been found is compared to the combinations present in the list. If there is a combination that is either better or worse than the new one, the better of them is kept in the list and the worse is discarded; otherwise, the new combination is added to the list. A combination is considered better than another one if the value of at least one parameter is closer to the original value and the values of the remaining parameters are equal.

After finishing the search, the system presents the list of combinations to the user. When the user clicks on some list element, the system re-clusters the frozen event subset using these settings and updates the snapshot map to present the results. By testing in this way the proposed variants, the user may choose the variant leading to best results, according to user's background knowledge and understanding of the character and properties of the event stream. Then, the chosen variant is passed to the main instance of the clustering tool.

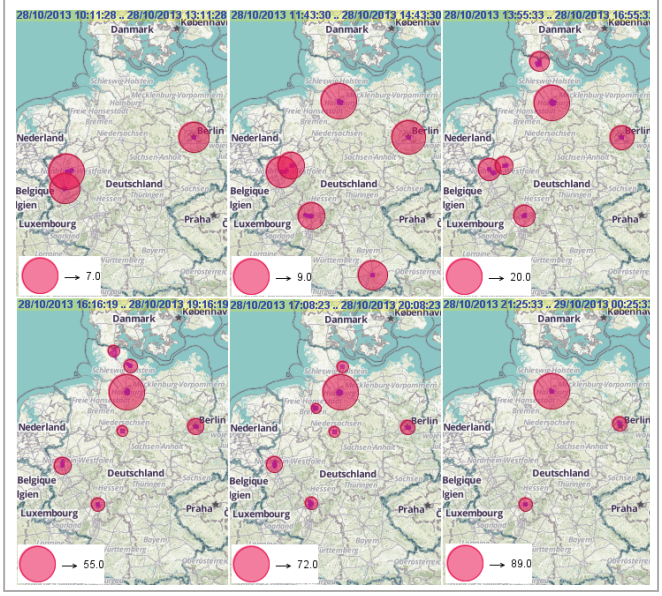
V. Case Studies

We have selected a set of case studies that can demonstrate the capability of our method to process event streams of various densities and to detect event clusters with different properties and behaviors over time. For testing the clustering and visualization tools, we have developed a simulator of a real time event stream, which takes events from a database and sends them to the clustering tool, applying a user-given scale factor to the time intervals between the events. The latter allows us to speed up the stream when necessary.

A. Tweets related to extreme weather conditions

We use a set of georeferenced Twitter messages posted on the territory of Germany on October 27-29, 2013 in relation to the storm Christian, a.k.a. St. Jude storm (http://en.wikipedia.org/wiki/St._Jude_storm). From all tweets posted in Germany during these days, potentially relevant ones were selected by finding occurrences of keywords indicating extreme weather conditions. Only 491 events have been found; hence, the event stream is sparse in both space and time. Generally, the Twitter users in Germany post much fewer messages with location coordinates than users in other countries. Even the whole stream of georeferenced tweets from Germany is quite sparse.

Fig. 5. Clusters of storm-related tweets in Germany on 28/10/2013.



For this case study, ground truth is available from the mass media and own experiences of some of the paper authors. This allowed us to check if the detected clusters were located in the affected areas and if the existence times of the clusters correspond to the times when the storm reached these areas.

Due to the large spatial and temporal scales of the analysis and the sparseness of the event stream, we use the following parameter settings: R = 25 km, ΔT = 3 hours, k = 2, and N = 5. We set an additional significance constraint: a cluster must contain events from at least two distinct Twitter users. We run the tool and see that nothing significant happened in Germany on 27/10/2013, except for a small cluster of tweets in Berlin. We interactively access the tweet texts and see that they either occasionally contain some of the keywords or refer to events occurring in the UK and the Netherlands.

In Fig. 5, there are 6 screenshots of the situation display corresponding to different 3-hour time intervals on 28/10/2013, when the storm reached Germany. The red circles show the current positions of the cluster centers and the total numbers of events in the clusters. In the lower left corner of each map we have put a fragment of the map legend showing the number of tweets represented by the largest circle size. Please note how this number increases over time. On 28/10/2013, again, a cluster emerged in Berlin, but the messages did not really refer to the situation in the city. A similar cluster appeared in Munich. The cluster in Frankfurt contained mostly messages concerning flight delays due to the storm or turbulences during landing at the airport. The remaining clusters and their evolution correspond to the path of the storm over Germany: it first hit the west of Germany and then moved northeast to Schleswig Holstein, Lower Saxony, and Hamburg, where the largest cluster was detected. On the next day, the storm was over. A single cluster emerged in Hamburg with tweets referring to the yesterday's storm.

This case study shows the ability of our method to detect significant clusters in a sparse event stream. It also shows the

necessity to exercise caution when trying to use georeferenced social media for locating extreme events or affected areas. We are aware of successful examples of correlating positions of Twitter posts with earthquake epicenter locations and hurricane trajectories [10][11]. Still, our studies show that the positions of posts may sometimes be misleading since the users can react to events happening elsewhere. In our case, many storm-related tweets were posted in Berlin, which lied aside of the storm path. It should also be taken into account that clusters of posts are more likely to occur in highly populated places. The sizes of the clusters may correlate with the population and may not be indicative of the severity of the local conditions. We do not want to say that social media cannot be used for detecting and analyzing abnormal events, but we argue for the necessity of checking any results of computational processing by a human analyst. Interactive visual tools greatly facilitate this activity.

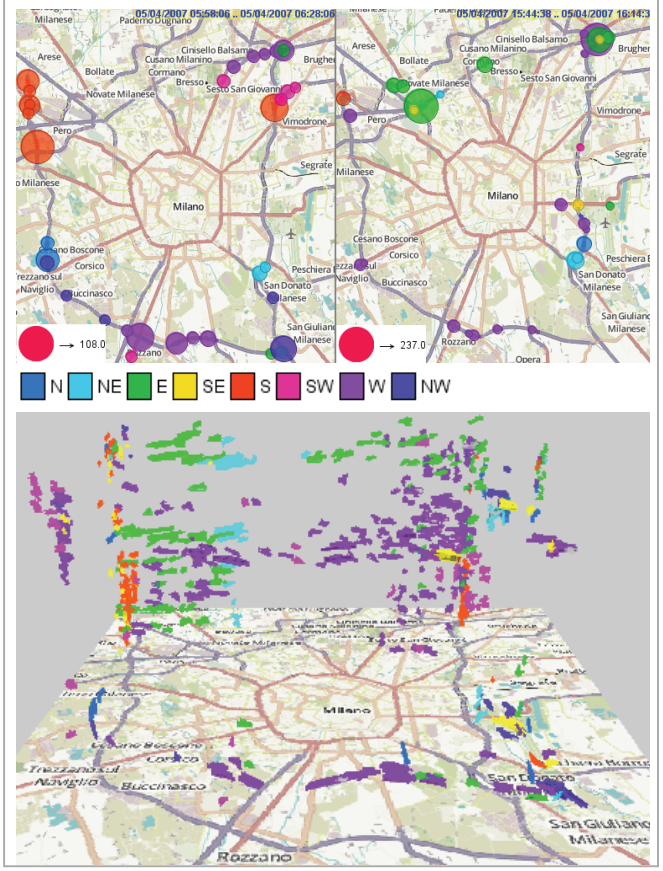
B. *Low speed events of cars*

From GPS tracks of 17,241 cars collected in Milan (Italy) during one week, we extracted the points where the speed was below 10 km/h but positive, i.e., stops were ignored. With these points, we simulated a stream of low speed events as if they were emitted by the cars in real time. In the following example, we use a one-day subset of the events. The event stream is quite dense, with the total of 100,994 events in one day and up to 184 events per minute. For this kind of data, one can expect that event clusters have linear shapes aligned to streets and that they may expand and/or move along the streets. Our experiments showed that our method can successfully process a stream of high density and detect event clusters of expected shapes. In fact, as we applied a time scale factor of 0.0005 to 0.0001, the experiments showed that the tool can cope with real event streams of much higher temporal density.

The aim of the supposed monitoring scenario is to detect and track traffic jams indicated by spatio-temporal clusters of low speed events coming from multiple cars. On the same street segment, there may be a jam in one direction but free movement in the other. The movement directions of the cars need to be accounted for and presented to the observer. According to the typical spatial and temporal scales of the phenomena to be observed (i.e., traffic jams), we choose the following parameters: R = 150 m, ΔT = 10 minutes, k = 1, N = 10, the number of distinct event sources (i.e., cars) is at least 3, and the difference between the movement directions of events to be put in the same cluster is at most 20°. The value 1 for the connectivity threshold is chosen since it can be expected that, if two event circles have one connecting event, the movement directions of their member events are close, and, moreover, they are likely to lie on the same street. It is therefore reasonable to consider them as parts of the same traffic jam.

Fig. 6 shows how our method clustered this event stream. Two example screenshots of the map display refer to the morning (left) and afternoon (right). In the lower left corners of the map images, fragments of the map legend explain the circle sizes. The sizes are proportional to the numbers of events in the clusters, being indicative of the severity of the traffic jams. The circles are additionally differentiated by colors representing the movement directions. A color legend is available below the map images. The convex hulls of the clusters are not

Fig. 6. Clusters of low speed events of cars in Milan on 05/04/2007. The colors of the circles represent the movement directions.Top: two screenshots of the map display referring to the morning (left) and afternoon (right). Bottom: the clusters from the whole day are represented in a space-time cube by their convex hulls colored according to the movement directions.



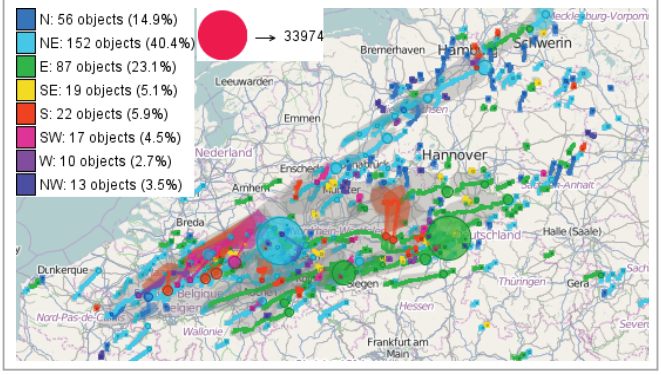
discernible on the map due to the small map scale and their narrow linear shapes, but the space-time cube at the bottom of Fig. 6 clearly shows that our method successfully finds clusters of linear shapes and can correctly group events according to values of thematic attributes, such as movement directions: it can be seen that the movement directions in the clusters correspond to the directions of the underlying streets.

C. *Lightning strikes*

A stream of lightning strike events was simulated based on real data collected by the company nowcast GmbH, Germany (www.nowcast.de). The purpose of the monitoring is to detect trends in the evolution of the thunderstorm. The example dataset consists of 138,466 lightning events that occurred in one day with the temporal density reaching the maximum of 467 events per minute. In this case study we tested, in particular, how our method can capture and represent moving clusters. The illustrations in Figs. 2, 3, and 7 show that the method successfully captures cluster movement as well as expansion and shrinking in space.

Fig. 7 shows all clusters of lightning strikes detected with the following parameter settings: R = 3.5 km, ΔT = 20 minutes, k = 3, N = 20. The trajectory lines show how the clusters moved. The lines and the circles representing the cluster sizes

Fig. 7. Movement of clusters of lightning strike events.



are colored according to the spatial directions of the vectors connecting the initial and final states of the clusters. We see that the thunderstorm started on the northeast of France, moved and expanded in the northeastern and eastern directions and, when being over the territory of Germany, divided into two currents moving to the northeast and to the east. As these movement trends are detected in real time, there is a principal possibility of making short-term predictions on this basis.

VI. Discussion

The case studies show that our method appropriately suites to the scenario and problem settings introduced in sections I and II.A2. None of the previously existing methods can do the required job. Our method has a unique capability to track cluster changes: spatial expansion and shrinking, splitting and merging, and directed movement. At the same time, our method is similar to density-based clustering in the capability to find dense concentrations of objects, as discussed in section III.G. To confirm this empirically, we have applied a DBC algorithm (OPTICS [4]) to the data from our case studies. This was done in a static manner, i.e., not in real time. For the DBC, we took R as the distance threshold, k+1 as the minimal density constraint, and considered only clusters with at least N events ($N \geq k$+1) as significant, i.e., the members of smaller clusters were re-classified as noise. We compared the proportions of the events included in clusters and in the noise by our method and by DBC. It is pointless to compare the numbers or the contents of the detected clusters, because our method merges and splits clusters, which does not happen in DBC.

Table 1 shows the statistics of putting events to clusters and to the noise by our method and by OPTICS. Almost all events that were put to noise by OPTICS were also put to noise by our algorithm. For the events put in clusters, the correspondence is very high for the case studies 1 and 3 and slightly lower for case study 2. The latter can be explained by diverse ways of dealing with the movement directions of the events. DBC attached an event to an existing cluster when its direction was close to that of just one neighboring event while our method did this when the direction was close to the average direction in an event circle, which is a stricter condition. Overall, taking OPTICS as a benchmark, we conclude that our method can find dense clusters well enough. The empirical results are consistent with the reasoning presented in section III.G.

TABLE I. Comparison of Our Method with OPTICS

Measure	Method	CS 1	CS 2	CS 3
% of events put in clusters	OPTICS	43.8	15.7	94.4
	our	45.3	13.6	89.1
	both	42.3	13.4	88.9
Ratio ours/OPTICS		**1.03**	**0.87**	**0.94**
Ratio both/OPTICS		**0.97**	**0.85**	**0.94**
% of events put in noise	OPTICS	56.2	84.3	5.6
	our	54.7	86.3	10.9
	both	53.2	84.0	5.5
Ratio ours/OPTICS		**0.97**	**1.02**	**1.95**
Ratio both/OPTICS		**0.95**	**0.99**	**0.98**

VII. Conclusion

We have developed a novel incremental clustering method for real-time detection and tracking of significant (i.e., sufficiently dense and large) clusters in streams of spatial events. The effectiveness and efficiently of the method has been tested and confirmed in several case studies with real data. In the capability to find dense concentrations of events, our method is similar to density-based clustering. A unique capability of our method is tracking cluster evolution over time. The algorithm is implemented in Java and integrated in a visual analytics environment that supports monitoring of the event stream and analysis of trends in its development. In the future, we shall work on finding the ways to use method results for short-term prediction in real time.

References

[1] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, "A framework for clustering evolving data streams", in: Proc. 29th Int. Conf. Very Large Data Bases, Berlin, Germany, 2003.

[2] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise", in: Proc. 6th SIAM Int. Conf. Data Mining, SIAM, Bethesda, Maryland, USA, 2006.

[3] M. Ester, H.-P.,Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", in: Proc. ACM KDD 1996, pp. 226-231, 1996.

[4] M.Ankerst, M.Breunig, H.-P.Kriegel, and J.Sander, "OPTICS: Ordering points to identify the clustering structure", in: ACM SIGMOD 1999, pp. 49–60.

[5] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data", in: Advances in spatial and temporal databases, 9th Int. Symp. SSTD 2005, Springer, pp.364-381, 2005.

[6] I. Peca, G. Fuchs, K. Vrotsou, N. Andrienko, and G. Andrienko, "Scalable Cluster Analysis of Spatial Events", in: Proc. EuroVA, Int.Workshop on Visual Analytics, EuroGraphics, pp.19-23, 2012.

[7] D. Thom, H. Bosch, S. Koch, M. Wörner, and T. Ertl, "Spatiotemporal anomaly detection through visual analysis of geolocated twitter messages", in: Proc. IEEE Pacific Visualization Symp., 41-48, 2012.

[8] M.-R. Bouguelia, Y. Belaïd, and A. Belaïd, "An Adaptive Incremental Clustering Method Based on the Growing Neural Gas Algorithm", in: Proc. 2nd Int. Conf. Pattern Recognition Applications and Methods - ICPRAM 2013, 42-49, 2013.

[9] N. Andrienko and G. Andrienko, "Spatial generalization and aggregation of massive movement data", IEEE Trans. Visualization and Computer Graphics, 17(2): 205-219, 2011.

[10] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes twitter users: real-time event detection by social sensors", in: Proc. 19th Intl. ACM Conf. Worldwide web, WWW '10, 851-860, 2010.

[11] Y. Kryvasheyeu, H. Chen, E. Moro, P. Van Hentenryck, and M. Cebrian "Performance of Social Network Sensors during Hurricane Sandy". PLoS ONE 10(2): e0117288. doi: 10.1371/journal.pone.0117288, 2015.