International Workshop on Visual Analytics (2012)
K. Matkovic and G. Santucci (Editors)

Scalable Cluster Analysis of Spatial Events

I. Peca[1], G. Fuchs[1], K. Vrotsou[1,2], N. Andrienko[1] & G. Andrienko[1]

[1] University of Bonn and Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), Germany
[2] Linköping University, Sweden

Abstract
Clustering of massive data is an important analysis tool but also challenging since the data often does not fit in RAM. Many clustering algorithms are thus severely memory-bound. This paper proposes a deterministic density clustering algorithm based on DBSCAN that allows to discover arbitrary shaped clusters of spatio-temporal events that (1) achieves scalability to very large datasets not fitting in RAM and (2) exhibits significant execution time improvements for processing the full dataset compared to plain DBSCAN. The proposed algorithm's integration with interactive visualization methods allows for visual inspection of clustering results in the context of the analysis task; several alternatives are discussed by means of an application example about traffic data analysis.

Categories and Subject Descriptors (according to ACM CCS): I.5.3 [Pattern Recognition]: Clustering—Algorithms

1. Introduction

Visual analytics strives to combine automated processing tools in a tight loop with human analyst interaction and feedback. This allows to utilize an analyst's intrinsic domain knowledge in order to appropriately choose and parametrize computational methods, evaluate intermediate results and guide the overall analysis process. In this regard, clustering is an indispensable tool for the analysis of large datasets. It can be used as a means for aggregating data to manageable sizes, removing noise and outliers, and consequently detecting structure and/or patterns in the data. A serious drawback of purely computational cluster analysis, however, is that the chosen cluster algorithm and set parameters are a 'black box' with respect to the outcome. Visualization can play a key role in helping the analyst to understand the impact of algorithm choice and parameter settings on the result.

We consider the discovery and visualization of arbitrary shaped clusters of events in space and time. An *event* is defined as any physical or abstract discrete object having a particular position in space and time. It can be instantaneous (point event) or it can have a temporal duration (time line). Examples of events include geo-referenced photos, occurrences of earthquakes, forest fires, disease cases, and mobile phone calls.

Clustering datasets of spatiotemporal events is challenging due to their typically vast size, which is often too large to fit into main memory, and due to the nature of the data which are inherently restricted by their spatiotemporal properties [AA09]. Partitioning clustering techniques, such as K-means, are avoided because the number of clusters must be known in advance, and also the retrieved clusters are restricted to convex shapes. Density based clustering algorithms, such as DBSCAN [EKSX96], are preferred for their ability to extract arbitrarily shaped clusters without requiring a priori specification of the cluster count. DBSCAN, however, operates with the entire dataset in main memory, and does not treat space and time differently.

We present a deterministic clustering algorithm based on DBSCAN that introduces critical improvements addressing these challenges. The algorithm

- is scalable to large datasets, i.e. data not fitting in RAM,
- accounts for the spatiotemporal nature of the data, and
- exhibits significant improvements in execution times.

Our approach is applicable to point events as well as events extended in space and/or in time assuming a meaningful definition of spatial and temporal distances between events.

2. Related work

Density-based approaches are a popular class of clustering algorithms, with DBSCAN [EKSX96, ABKS99] as the initial representative. DBSCAN is designed to discover clusters of arbitrary shape by considering the density of objects

DOI: 10.2312/PE/EuroVAST/EuroVA12/019-023

(i.e., points in feature space) according to a user-defined distance metric d and distance threshold ε_{space}. The *neighbourhood* of a point p is composed of the set of points $\{N_p | d(p, n_i) \leq \varepsilon_{space}\}$. If $|N_p| \geq minPts$, with $minPts$ a given neighbourhood size threshold, then p is a so-called *core point*; otherwise it is considered a *marginal point*. A point p is *density-reachable* from another point q iff p is part of the neighbourhood of q and q is a core object. DBSCAN starts with an arbitrary point p, and all points that are density-reachable from p are retrieved. If p is a core point, this procedure yields a cluster. If p is a marginal point, the algorithm proceeds to the next unvisited point in the database.

Clustering methods are typically memory-bound with respect to the maximum data size that can be processed. Several scalability optimizations have therefore been proposed, most often based on sampling. In [BKKK04] a *preference subspace* is defined by selecting a subset of dimensions for each point. '*Sampling inside* DBSCAN' [ZAC*00] does not save core-objects that have only cores as neighbours. This approach, however, may lead to *lost points* which have to be treated separately. '*Sampling outside* DBSCAN' [ZAC*00] selects a sample of the dataset and builds separate R*-trees for both the database and the sample. While DBSCAN is applied only to the sample, the cluster labeling process is applied to the entire database. However sampling techniques rely on the appropriate choice of an efficient sample, and must separately deal with *lost points*. To avoid this we do not use sampling in our approach.

Partitioning is also employed for improving the clustering process. In [ZSC*00] the data are partitioned according to their distribution characteristics. The partitions are clustered using DBSCAN, and the partial clusters are merged. This merging process is challenging when partitioning over multiple dimensions. We address this by ordering the data along one dimension and processing them in contiguous blocks.

ST-DBSCAN [BK06] is a method similar to our own. It uses two distance metrics, one for spatial and another for non-spatial object attributes including time. To speed up point neighbourhood queries temporal filtering (i.e., sampling) is applied; moreover the paper makes very specific assumptions about the temporal structure of the data, namely, temporal neighbourhood is defined based on cyclic time units (days, years). By contrast, our approach can handle arbitrary types of temporal references.

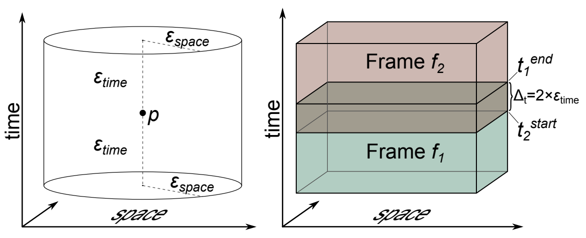
3. Approach

The algorithm we propose is an extension of DBSCAN inspired by the event clustering idea presented in [AAH*11] and ST-DBSCAN. We adopt the existing DBSCAN algorithm to larger-than-RAM spatio-temporal datasets by introducing a temporal window $[t_1, t_2]$ which allows us to consider smaller subsets of the data at a time. Contiguous blocks of data (or *frames*), composed of objects contained in the time window, are loaded in RAM and processed separately by the algorithm. The algorithm works as follows:

1. The data are loaded into RAM successively in frames which are partly overlapping (Section 3.2).
2. DBSCAN is applied to each frame using separate distance measures for space and time (Section 3.1).
3. When clustering is complete, a process is initiated for merging clusters of neighbouring frames (Secion 3.3).
4. After merging, the RAM occupied by old frames is released and the process is repeated for subsequent frames.

The parameters needed for the proposed algorithm are:

- The spatial, ε_{space}, and temporal, ε_{time}, density thresholds set by the user to define the neighbourhood (Fig. 1(a)).
- The neighbourhood size threshold $minPts$ defining the minimum number of points that must exist within ε_{space} and ε_{time} for a point to be considered a core point.
- The time window, $frameSize$, defining the size of each frame in terms of time units, days/hours etc. This is either set to a fixed value by the user (with a default value of $4 \cdot \varepsilon_{time}$), or it may be dynamically computed for each frame with respect to temporal data density and available RAM.



(a) Neighbourhood (b) Data frame overlap

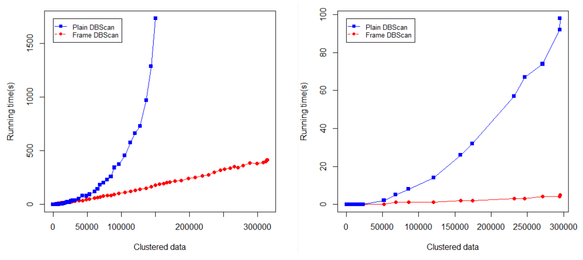
Figure 1: *Schematic views of algorithm concepts. (a) The cylindrical neighbourhood of a point p in the space-time domain. (b) The data frame overlap in the temporal domain.*

3.1. Distance measures

We use two distance measures, a spatial and a temporal, in the computation of our proposed algorithm. Spatial distance, d_s, is measured by the Euclidean distance of two objects. Temporal distance for linear time is computed by $d_t(t_1, t_2) = |t_1 - t_2|$, where t_1 and t_2 are the time interval boundaries for two objects. Similarly, time distance can be defined for cyclic time units.

3.2. Selection of frames

Before applying the algorithm, points are sorted in the database with respect to the temporal attribute and then processed in frames of chronological order. Temporal frame boundaries are selected such that a temporal overlap $\Delta_t = 2 \cdot \varepsilon_{time}$ between each pair of subsequent frames is introduced (Fig. 1(b)). This ensures that all core points (with a



(a) $\varepsilon_{space} = 500$, $minPts = 15$, $\varepsilon_{time} = 5$, $frameSize = 30$

(b) $\varepsilon_{space} = 1000$, $minPts = 5$, $\varepsilon_{time} = 10$, $frameSize = 50$

Figure 2: *Experiment results showing the execution times of both plain DBSCAN (blue) and our proposed algorithm (red) for processing the full dataset for two different representative parameter combinations.*

temporal neighbourhood size of $2 \cdot \varepsilon_{time}$) in the overlap region are indeed detected in both frames so the corresponding partial clusters are correctly merged in the next step (Section 3.3). It follows that $frameSize \geq 2 \cdot \varepsilon_{time}$ should hold. Otherwise, frames would overlap with more than one subsequent frame which results in an excessive number of merge operations, significantly degrading algorithm performance.

3.3. Merging process

DBSCAN is applied to each data frame separately. Once clusters of two adjacent frames have been found, they are inspected and merged if needed. There are two cases that cover all possible situations for merging the clusters. Two clusters from adjacent frames are merged if:

1. they share a core point in the overlapping area, or
2. they share a point that is core in one cluster, and marginal in a core's neighbourhood in the other.

Clusters without points in the overlapping area are ignored. Cluster merging is done by updating the corresponding labels: objects in the second frame are relabeled to match the cluster id from the first frame. Using this approach the algorithm revisits each object at most once.

3.4. Optimizations

To speed up the retrieval of an object's neighbourhood during clustering we maintain one indexing tree per frame, specifically, an optimized R-tree using the Sort-Tile-Recursive (STR) algorithm [RSV01]. Since the data points are already sorted, it is possible to quickly generate an R-tree with high space utilization for each frame separately.

Furthermore, in order to efficiently merge clusters from different frames, we employ a cluster label-object dictionary structure: we first identify the cluster label to be changed, and then bulk update the actual objects associated with that cluster label. This speeds up the merging considerably, since the number of clusters is smaller than the number of objects.

3.5. Performance, complexity and quality

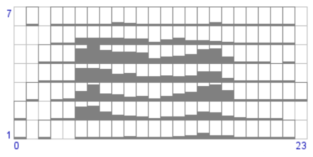
The computational complexity of the proposed method is primarily resulting from the DBSCAN algorithm. For each of the n entries in the dataset, their ε_{space} and ε_{time} neighbourhood need to be retrieved. This can be done in $O(n \cdot log(n))$ using indexing tree structure over the *spatial dimension*. In addition, the dataset is divided in k frames, each with size n_i. The time complexity of the algorithm results from querying the neighbours of each object which is performed locally within each frame. The complexity is thus $O(\sum_{i=1}^{k} n_i log(n_i))$, where k is the number of frames, and n_i is the number of points in each frame. The computational complexity is $O(k \cdot n_i \cdot log(n_i)) = O(n \cdot log(n_i))$, where $n_i \ll n$. Thus, our proposed method often outperforms plain DBSCAN even if the entire data would fit into RAM – depending on the organization of the data, the faster neighbourhood queries over individual frames outweighs the costs for the chronological ordering preprocess.

Several experiments were conducted to compare the behaviour of the original DBSCAN algorithm and our modified version. The experiments, given varying parameters, investigated differences in computation time as data size increases up to and beyond the maximum size fitting into RAM. Two datasets were used for the experiments consisting of around 300,000 points each. The experiments were conducted on an Intel® Core^{TM} i3 CPU 4GB RAM 64-bit OS.

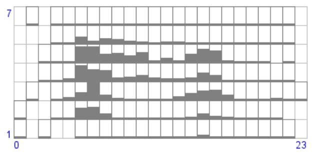
Experiment results (Fig. 2) show that our approach outperforms original DBSCAN for large datasets. Computation times increase for both algorithms with data size, however the maximum size supported by our algorithm is far larger. The experiments indicate that the optimal size of the time window is $2 \cdot \varepsilon_{time} < frameSize < 5 \cdot \varepsilon_{time}$. Larger frames offer diminishing returns since defeating the purpose of *frames*, while $2 \cdot \varepsilon_{time}$ is the minimum size threshold that ensures all core points are properly extracted to match plain DBSCAN results after merging of frames (cf. Sections 3.2, 3.3). Note that provided $frameSize \geq 2 \cdot \varepsilon_{time}$, the impact of parameter combinations for ε_{space}, ε_{time} and *minPts*, as well as clustering results exactly match those of plain DBSCAN.

4. Visual exploration of clustering results

We will illustrate how the proposed clustering algorithm can be used for visual analysis of large datasets. For our example, we use a dataset of 17,200 GPS-tracks of cars in Milan collected over one week (~2 million position records). Our objective is to detect traffic jams in the city during this period and investigate their properties. Traffic jams can be identified through spatio-temporal clusters of slow movement events. Using database operations, we extract position records with speed less than 10km/h (295,000 records). We then apply our clustering algorithm with parameters: $\varepsilon_{space} = 100m$, $\varepsilon_{time} =$

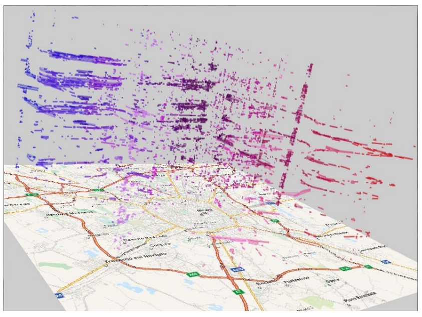


(a) Number of clusters



(b) Number of events in clusters

Figure 3: *2D histograms showing aggregates of traffic jams with respect to days (vert. axis) and hours of day (horiz. axis).*


(a) Clusters of entire week

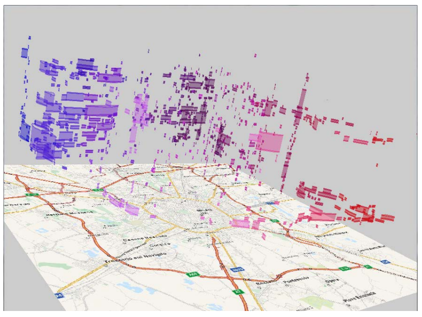

(b) Clusters of single day (Thursday)

Figure 4: *Clusters represented in a space-time cube by prisms enclosing them with respect to their start and end times, and durations. The prisms are coloured according to spatial position: northwest is blue, northeast is purple, southeast is red, southwest is pink.*

$10min$, $minPts = 5$, and $frameSize = 50$. The method finds 6,166 clusters including in total 75,691 points. Each cluster is characterized by the number of events in it, its duration, and start and end time. The durations of the clusters range from 34 seconds to 242 minutes, 43% of them have duration up to 10 minutes while very long clusters are rare.

We interactively filter out clusters with durations below 10 minutes and consider the remaining 3,513 clusters as representing traffic jams. We investigate the temporal distribution of these traffic jams by means of two-dimensional (2D) histograms with the dimensions representing the days from 1 (Sunday) to 7 (Saturday) and hours of the day from 0 to 23 (Fig. 3). The heights of the bars show counts of traffic jams (Fig. 3(a)), and counts of events in the traffic jams (Fig. 3(b)). As could be expected, traffic jams are more frequent on working days rather than weekends. The count of traffic jams peaks around hours 6 and 16. The number of events in traffic jams peaks around 5-6h and 15-16h.

To enable further investigation of the detected traffic jams they can be displayed in a spatial context by convex hulls enclosing the clusters: on a map by drawing 2D polygons around the cluster events, or in a space-time cube (STC) [Häg70, Kra03] by prisms around the clusters with respect to the clusters' start and end times, and durations (Fig. 4). Figure 4(a) displays traffic jams of the entire week in the STC coloured according to their spatial position. Traffic jams occur frequently on the northwest and northeast part of the city. Also, clusters with long duration occur mostly on the east, close to Linate airport. These, however, are probably not traffic jams but usual slow movements in the airport area. Spatial and/or temporal zooming can be applied, to study clusters in more detail. In Figure 4(b) the STC has been temporally zoomed to show a single day, specifically, day 5 (Thursday) when most traffic jams occurred. Traffic jams on the northwest occurred continuously during the day, while on the southeast they occurred for short intervals in the morning and in the afternoon. Also, very long (in time) traffic jams are revealed in the northern part of the eastern belt road in the morning and on the southern belt road in the afternoon. Interaction with the STC gives access to additional information about selected clusters allowing further exploration and analysis.

As further informal verification of the proposed algorithm we cross-compared the extracted traffic jams with the results obtained using the clustering method described in [AAH*11] and found them to be the same.

5. Conclusion and Future Work

We proposed a deterministic event clustering algorithm based on DBSCAN that is scalable to very large data not fitting into main memory. The core idea is to sequentially cluster temporal frames (subsets) of the data and then merge them to obtain the final result. Evaluation shows a significant performance increase in comparison to plain DBSCAN for both data fitting in RAM and data too large to fit.

We further discussed how our proposed method is integrated with interactive visualization methods to facilitate evaluation of parametrization choices and visual exploration of clustering results. This combination of density-based clustering and interactive result refinement allowed us to find interesting traffic jam patterns in a car movement dataset.

Currently, our proposed framing scheme only considers temporal frames. Clustering is thus still memory-bound in cases where the minimum temporal frame size is still too large to fit into RAM. We thus plan to investigate combining temporal and spatial partitioning with dynamic frame sizes for even better scalability to data with different spatial vs. temporal density distributions.

References

[AA09] ANDRIENKO G., ANDRIENKO N.: Interactive cluster analysis of diverse types of spatiotemporal data. *SIGKDD Explorations 11*, 2 (2009), 19–28. 1

[AAH*11] ANDRIENKO G., ANDRIENKO N., HURTER C., RINZIVILLO S., WROBEL S.: From movement tracks through events to places: Extracting and characterizing significant places from mobility data. In *IEEE Visual Analytics Science and Technology* (2011), IEEE, pp. 161–170. 2, 4

[ABKS99] ANKERST M., BREUNIG M., KRIEGEL H., SANDER J.: Optics: Ordering points to identify the clustering structure. In *ACM SIGMOD'99 Int'l Conf. on Management of Data* (1999), ACM Press, pp. 49–60. 1

[BK06] BIRANT D., KUT A.: St-dbscan: An algorithm for clustering spatial-temporal data. *Data & Knowledge Engineering 60* (2006), 208–221. 2

[BKKK04] BÖHM C., KAILING K., KRIEGEL H., KRÖGER P.: Density connected clustering with local subspace preferences. In *Fourth IEEE International Conference on Data Mining* (Nov. 2004), IEEE Computer Society Press, pp. 27–34. 2

[EKSX96] ESTER M., KRIEGEL H., SANDER J., XU X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In *Int'l Conf. on Knowledge Discovery and Data Mining* (1996), AAAI Press, pp. 226–231. 1

[Häg70] HÄGERSTRAND T.: What about people in regional science? *Papers in Regional Science 24* (1970), 6–21. 4

[Kra03] KRAAK M.: The space-time cube revisited from a geovisualization perspective. In *Proceedings of the 21st International Cartographic Conference (ICC)* (Aug. 2003), pp. 1988–1996. 4

[RSV01] RIGAUX P., SCHOLL M., VOISARD A.: *Spatial Databases: With Application to GIS*. Morgan Kaufmann, 2001, ch. Spatial Access Methods, pp. 201–266. 3

[ZAC*00] ZHOU S., A.ZHOU, CAO J., WEN J., FAN Y., HU Y.: Combining sampling technique with DBSCAN algorithm for clustering large databases. In *4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications* (2000), Springer-Verlag, pp. 169 – 172. 2

[ZSC*00] ZHOU A., S.ZHOU, CAO J., FAN Y., HU Y.: Approaches for scaling DBSCAN algorithm to large spatial databases. *Computer Science and Technology. 15*, 6 (2000), 509–526. 2