Industrial and Applications Paper

In-DBMS Sampling-based Sub-trajectory Clustering

Nikos Pelekis
Dept. of Statistics and Ins. Science
University of Piraeus
Piraeus, Greece
npelekis@unipi.gr

Panagiotis Tampakis
Dept. of Informatics
University of Piraeus
Piraeus, Greece
ptampak@unipi.gr

Marios Vodas
Dept. of Informatics
University of Piraeus
Piraeus, Greece
mvodas@unipi.gr

Costas Panagiotakis
Dept. of Business Administration
TEI of Crete
Agios Nikolaos, Crete, Greece
cpanag@staff.teicrete.gr

Yannis Theodoridis
Dept. of Informatics
University of Piraeus
Piraeus, Greece
ytheod@unipi.gr

ABSTRACT

In this paper, we propose an efficient in-DBMS solution for the problem of sub-trajectory clustering and outlier detection in large moving object datasets. The method relies on a two-phase process: a voting-and-segmentation phase that segments trajectories according to a local density criterion and trajectory similarity criteria, followed by a sampling-and-clustering phase that selects the most representative sub-trajectories to be used as seeds for the clustering process. Our proposal, called S^2T-Clustering (for Sampling-based Sub-Trajectory Clustering) is novel since it is the first, to our knowledge, that addresses the pure spatiotemporal sub-trajectory clustering and outlier detection problem in a real-world setting (by 'pure' we mean that the entire spatiotemporal information of trajectories is taken into consideration). Moreover, our proposal can be efficiently registered as a database query operator in the context of extensible DBMS (namely, PostgreSQL in our current implementation). The effectiveness and the efficiency of the proposed algorithm are experimentally validated over synthetic and real-world trajectory datasets, demonstrating that S^2T-Clustering outperforms an off-the-shelf in-DBMS solution using PostGIS by several orders of magnitude.

CCS Concepts

• Information systems → Information systems applications → Data mining → Clustering • Information systems → Information systems applications → Spatio-temporal systems

Keywords

Mobility data mining; Sub-trajectory clustering; Trajectory segmentation; Trajectory sampling; MOD engines

1. INTRODUCTION

Knowledge discovery in mobility data [11][29][46][42] exposes patterns of moving objects exploitable in several fields. For instance, in both mature (transportation, climatology, zoology, etc.) and emerging domains (e.g. mobile social networks), scientists work with mobility-aware (mostly GPS-based) data, resulting in trajectories of moving objects stored in Moving Object Databases (MOD). Although during the recent years, there have been made significant achievements in the field [11][29][46][42], ongoing research calls for new methods aiming at deeper comprehension and analysis of mobility. For instance – and acting as motivation of this work – enhancing MOD engines, such as Secondo [1] and Hermes [31], with data mining operators is challenging [11][29] and is subject to the indexing extensibility interface of the corresponding ORDBMS on which they are implemented (see GiST [14][20], for example).

In the literature of trajectory-based mobility data mining, one can identify several types of mining models used to describe various collective behavioral patterns. As such, there exist works that identify various types of clusters of moving objects [10][26][21][32] and variations [4][17][22][44]. Related line of research is the one that builds representatives out of a trajectory dataset, either by generating artificial data [21][32] or by sampling the dataset itself [33][28].

Focusing on trajectory clustering, the majority of related work proposes a variety of distance functions, utilized by well-known clustering algorithms to identify collective behavior among whole trajectories [26][32][30]. A parallel line of research tries to discover local patterns in MOD, i.e. patterns that are alive only for a portion of moving objects' lifespan: some of those techniques simplify the given trajectories, however focusing on the spatial and ignoring the temporal dimension, such as TRACLUS [21], which is considered as the current state-of-the-art sub-trajectory clustering technique.

Figure 1 illustrates a working example that motivates our research: a dataset consisting of four trajectories, T_1, …, T_4. (In this figure, the time dimension is ignored for visualization reasons.) Among the sub-trajectories that compose the dataset, our goal is to identify two clusters (in red and blue, respectively) and five outliers (in black). In particular, the first (red) cluster consists of the tails of trajectories T_1, T_2 and T_3, the second (blue) cluster consists of the main bodies of trajectories of trajectories T_1, T_2, T_3 and T_4, while the rest portions of the trajectories (namely, the tail of T_4 and all four heads) are recognized as outliers.

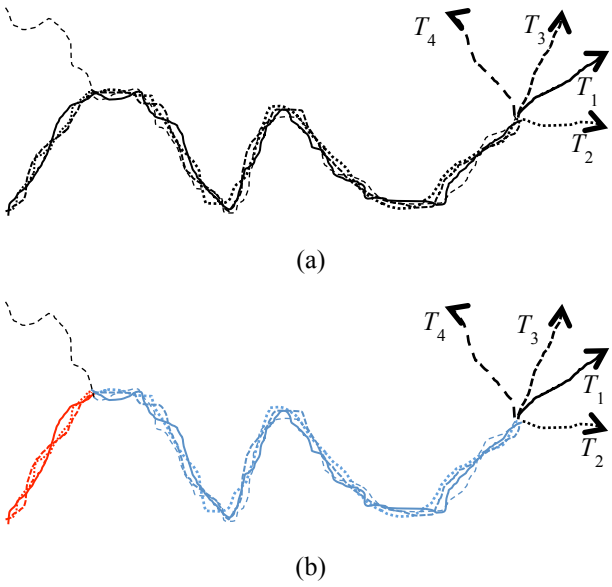


Figure 1. (a) a MOD of 4 trajectories; (b) the MOD split in 2 clusters (in red and blue) and 5 outliers (in black).

Such clustering sounds impossible to be achieved by TRACLUS. This is due to the inherent design of that algorithm that, as delineated by the authors, discovers linear patterns only and fails to identify complex (e.g. snake-like) patterns like the ones that appear in Figure 1. In other words, when applied to this dataset, TRACLUS would eventually discover five to six linear clusters (one new cluster each time the snake-like motion changes direction). On the contrary, we wish to be able to follow these direction changes without assuming underlying constraints on the complexity of the shape of sub-trajectories found nor posing geometrical and temporal constraints, in terms of algorithm parameters, as those required by related work, e.g. [4][17]. For those having experimented with those techniques, parameters like disc radius, minimum duration and cardinality of patterns, are hard to be set in advance. For instance, a small detour of an object belonging to one of the clusters, would probably result in either the lack of those patterns or the formation of smaller ones.

Inspired by the above, in this paper we study an important problem in the mobility data management and exploration domain [29], that of sub-trajectory clustering and outlier detection. Informally, we aim at a methodology that builds clusters around (and detects outliers far away from) appropriately selected sub-trajectories that preserve the properties and the mobility patterns hidden in a MOD, as much as possible. Towards this goal, we introduce a novel clustering methodology exploiting on the voting, segmentation and sampling concepts proposed in [28]. More specifically, we devise an efficient voting process that allows us to describe the 'representativeness' of a trajectory in a MOD as a smooth continuous descriptor [28]. Using these descriptors (their 'representativeness'), we result in the automatic segmentation of trajectories into 'homogenous' sub-trajectories. Next, a deterministic sampling procedure selects only those sub-trajectories that optimally describe the entire MOD. Finally, we devise a method for sub-trajectory clustering driven by the aforementioned representative sample of sub-trajectories.

The design of such a clustering methodology is subject to two indispensible requirements that challenged our research: we seek for (a) an efficient and scalable solution that (b) should be able to operate on a real-world DBMS rather than being an ad hoc implementation using a sophisticated access method. This is in order for the proposal to be practical and useful in real-world application scenarios, where concurrency and recovery issues are taken into consideration. Both requirements call for a MOD engine; therefore, our proposal is implemented as a query operator in Hermes [16], implemented on top of PostgreSQL. To our knowledge, it is the first time in the literature that GiST is used to index trajectory-based mobility data for the above purposes. Therefore, we argue that this is an important step towards bridging the gap between MOD management and mobility data mining, as state-of-art approaches [25][40][12] could make use of the efficiency and the advantage of our proposal to execute in-DBMS clustering via simple SQL.

Our contribution is summarized below:

- we formulate the problem of sub-trajectory clustering (and outlier detection) in a MOD as an optimization problem;
- we propose an efficient solution, the so-called S^2T-Clustering algorithm, driven by a deterministic sampling methodology, with the number of clusters being automatically detected by the algorithm;
- in order to speed up clustering tasks in MOD systems, we implement S^2T-Clustering as a query operator over an expensible DBMS, namely PostgreSQL, based on access methods that exploit on the GiST indexing extensibility interface. (For validation purposes, we also implement S^2T-Clustering using PostGIS, an off-the-shelf in-DBMS alternative solution.)

The rest of the paper is organized as follows: Section 2 presents related work and Section 3 formulates the problem of sub-trajectory clustering (and outlier detection). Sections 4 and 5 present our proposal and its in-DBMS realization, respectively. Experimental results that evaluate S^2T-Clustering using synthetic and real trajectory datasets from urban and vessel traffic domains are provided in Section 6. Section 7 concludes the paper.

2. RELATED WORK

During the past decade, the field of MOD has emerged as a strong candidate for the efficient management of trajectory data exploiting on the robust architecture of extensible DBMS; Secondo [1] and Hermes [31] are typical examples of this paradigm. Nevertheless, extending a DBMS does not reduce the complexity of understanding their concurrency and recovery protocols, and as such, does not reduce the implementation effort of an external access method when compared to a built-in one, assuming that identical levels of concurrency, robustness and integration are desired [20]. Actually, complexity is the main reason that almost none of the numerous access methods for mobility data that have been proposed in the literature, [34][36][13] to name but a few representatives, have been integrated in a real Object-Relational DBMS. Even GiST [14] that has been proposed to serve access method extensibility has not been used so far in the context of mobility data. Mainly due to the above reasons, although a lot of research has been carried out in the field of MOD regarding efficient indexing and query processing, almost no related work exists in the field of mobility data mining in-DBMS [29].

Focusing on plain (i.e. outside DBMS) implementations, the common building block of trajectory clustering approaches is the use of different similarity functions as the means to group trajectories into clusters. Such a similarity function is proposed in [8] for the efficient processing of most-similar trajectory (MST) queries. T-OPTICS [26] incorporates a similar distance function into the well-known OPTICS [3]. In [5], probabilistic techniques based on EM algorithm are proposed for clustering (short) trajectories using regression mixture models. In [32], the

authors propose CenTR-I-FCM, a variant of Fuzzy C-means (FCM) for MOD, while in [39] introduce the concept of uncertain group pattern. Both approaches propose specialized similarity functions having as goal to tackle the inherent uncertainty of trajectory data. In [8], the authors introduced the vector field k-means trajectory clustering technique whose central idea is to use vector fields to induce a notion of similarity between trajectories, letting the vector fields themselves define and represent each cluster. In [41], a multi-kernel-based estimation process leverages both multiple structural information within a trajectory and the local motion patterns across multiple trajectories in order to face challenges in case of large variations within a cluster and ambiguities across clusters. In [15], the Clustering and Aggregating Clues of Trajectories (CACT) pattern mining framework has been proposed for discovering trajectory routes that represent the frequent movement behaviors of a user. The approach exploits on a similarity measure for trajectories with silent durations (i.e., the time durations when no data points are available to describe the movements of users), which is used in a clue-aware clustering algorithm, where clues are some spatially and temporally close data points that capture certain common partial movement behaviors of the user.

TRACLUS [21] is a partition-and-group framework for clustering 2D trajectories (i.e. it ignores the time dimension), enabling the grouping of similar sub-trajectories, according to a trajectory partitioning step that uses the minimum description length principle. In its core, it uses a variant of DBSCAN [7], operating on the partitioned directed line segments. This work was the first to tackle the problem of identifying sub-patterns in trajectory data; however, it presents certain limitations (as discussed earlier) under the prism of the specifications we posed. In [24] the authors introduce an incremental trajectory clustering that exploits on TRACLUS.

Another line of research includes works that aim to discover several types of collective behavior among moving objects, forming a group of objects that moves together for a certain time period, such as moving clusters [18], flocks [4], convoys [17], swarms [23], traveling companion [36][37], gathering [44][45], and platoon [22] patterns. Although these approaches provide lucid definitions of the mined patterns, their main limitation is that they search for special collective behaviors, defined by respective parameters.

Our approach also finds commonalities to well-known approaches of clustering algorithms of point (vector) data [43][35], which sample the dataset at a pre-processing step and then perform the core clustering process aiming at high efficiency. However, these vector-based algorithms are not applicable to MOD due to the complex structure and properties of mobility data. Moreover, there is an essential difference between those techniques and our approach: while those mainly rely on random sampling, in our approach the clustering is driven by a sample resulted by an optimization formula, thus leading to a deterministic solution of the sub-trajectory clustering problem.

As already discussed, plain (sub-)trajectory clustering implementations leave concurrency and recovery outside the scene of requirements, as such setting limitations to their usage in real-world applications. In contrast, in this work we provide efficient in-DBMS solutions ready to be used by domain experts maintaining their volumes of data in state-of-the-art DBMS.

3. PROBLEM FORMULATION

Let $D = \{T_1, T_2, \ldots, T_N\}$ be a dataset consisting of N trajectories of moving objects (we assume that the objects move in the xy-plane). Let $p_{k,i} = (x_{k,i}, y_{k,i}, t_{k,i})$ be the i-th sampled point, $i \in \{1, 2, \ldots, L_k\}$ of trajectory T_k, $k \in \{1, 2, \ldots, N\}$, where L_k denotes the length of T_k (i.e. the number of points it consists of), the pair $(x_{k,i}, y_{k,i})$ and $t_{k,i}$ denote the 2D location and the time coordinate of point $p_{k,i}$, respectively. We consider linear interpolation between two successive sampled points, $p_{k,i}$ and $p_{k,i+1}$, so that each trajectory turns out to be a sequence of 3D line segments, $e_{k,i} = (p_{k,i}, p_{k,i+1})$, of cardinality $L_k - 1$, where each segment represents the continuous movement of the object during sampled points. Table 1 summarizes the definitions of the symbols used in this paper.

Table 1. Table of Symbols

Symbol	Definition
D	A dataset, $D = \{T_1, \ldots, T_N\}$, of N trajectories
T_k	k-th trajectory of D
$p_{k,i}$	i-th point of trajectory T_k, $p_{k,i} = (x_{k,i}, y_{k,i}, t_{k,i})$
L_k	Number of points forming trajectory T_k
$e_{k,i}$	i-th (3D) line segment of T_k, $e_{k,i} = (p_{k,i}, p_{k,i+1})$
LP_k	Number of sub-trajectories partitioning T_k
P_k	Set of the sub-trajectories partitioning T_k
$P_{k,i}$	$i-$th sub-trajectory of trajectory T_k
P	Set of sub-trajectories in dataset D, $P = \cup P_k$
V_k	Voting descriptor of trajectory T_k
V	Set of voting descriptors in dataset D, $V = \cup V_k$
$VP_{k,i}$	Voting descriptor of sub-trajectory $P_{k,i}$
$Nl_{k,i}$	Normalized lifespan descriptor of sub-trajectory $P_{k,i}$ w.r.t. lifespan of T_k
C	Clustering of sub-trajectories in M clusters, $C = \{C_1, \ldots, C_M\}$, $C_i \subset P$, $C_i \cap C_j = \emptyset$, $i \neq j$
S	Sampling set of representatives, $S = \{R_1, \ldots, R_M\}$, $S \subset P$, with sub-trajectory R_i representing cluster C_i
M	Cardinality of C (and S)
SR(S)	Representativeness function of S
$V(P_{k,i}, R_j)$	Voting descriptor of $P_{k,i} \in P-S$ w.r.t. sub-trajectory $R_j \in S$
Out	Set of outlier sub-trajectories, $Out = P-C$

Informally, the objective of sub-trajectory clustering is to partition trajectories into sub-trajectories and then form groups of similar ones, while at the same time, separating those that cannot fit in a group (called outliers). However, searching for entire trajectory similarity may be misleading since real-world trajectories may be long and consisting of heterogeneous portions of movement [6]. On the other hand, clustering at the sub-trajectory level sounds much more effective.

Rephrasing the previous discussion, if we consider trajectory T_k as a sequence of successive sub-trajectories $P_{k,i}$ of arbitrary length ($P_{k,i}$ is the i-th sub-trajectory of trajectory T_k), the objective of sub-trajectory clustering (and outlier detection) is to partition sub-trajectories into groups of similar ones and isolate the ones (called outliers) that are very dissimilar from the others. To achieve this, assuming a cluster is represented by its representative (or centroid) sub-trajectory, we define clustering as an optimization problem where the optimization criterion is to maximize the following expression:

$$SRD = \sum_{R_j \in S} \sum_{P_{k,i} \in C(R_j)} \overline{V(P_{k,i}, R_j)} \qquad (1)$$

The formula to be maximized, namely *Sum of Representativeness of Dataset* (SRD), uses set $S = \{R_1, \ldots, R_M\}$ of the representative sub-trajectories and the corresponding clusters $C(R_j)$ built around them, and is calculated upon $\overline{V(P_{k,i}, R_j)}$, i.e. the mean similarity (or average number of votes, according to our terminology) of sub-trajectory $P_{k,i}$ with respect to R_j.

Given the above formulation, the problem in hand is formalized as follows:

Problem 1 (Sub-Trajectory clustering in a MOD): *Assuming a dataset $D = \{T_1, T_2, \ldots, T_N\}$ consisting of N trajectories, where each of them is considered as a sequence P_k of successive sub-trajectories of arbitrary length, the problem of sub-trajectory clustering is defined as the task of partitioning the set $P = \cup P_k$ of sub-trajectories into (i) a clustering $C = \{C_1, \ldots, C_M\}$ of M clusters, $C_i \subset P$, $C_i \cap C_j = \emptyset$, $i \neq j$ (i.e. hard clustering), where each cluster is represented by its representative sub-trajectory $R_j \in P$, $j = 1, \ldots, M$, and (ii) a set Out of outliers, by maximizing Eq. (1).* ■

It is important to note that maximizing Eq. (1) is not trivial at all since one has to define, among others, (i) the criterion according to which a trajectory is segmented into sub-trajectories, (ii) the technique for selecting the set of the most representative sub-trajectories, (iii) whose cardinality M is unknown, to name but a few challenging sub-problems.

4. THE S^{2}T-CLUSTERING ALGORITHM

In this section, we propose a solution for Problem 1 defined above, which is called S^{2}T-Clustering (for Sampling-based Sub-Trajectory Clustering). Our proposal (listed in Algorithm 1) consists of two phases: first, we apply the so-called *Neighborhood-aware Trajectory Segmentation* (aka NaTS) method that is able to detect homogenized sub-trajectories applying trajectory voting and segmentation; then, we apply the so-called *Sampling, Clustering, and Outlier detection* (aka SaCO) method that selects the most representative among the sub-trajectories detected in the previous phase in order for them to serve as the seeds of the clusters to be produced.

Algorithm 1. S^{2}T-Clustering

Input: trajectory dataset $D = \{T_1, T_2, \ldots, T_N\}$, voting influence σ, threshold ε
Output: sampling set S, clustering C, set of outliers Out.

```
    // Initialization phase
1.  Reset set V of voting descriptors in D
    // NaTS phase (Neighborhood-aware Trajectory Segmentation)
2.  for each trajectory T_k ∈ D do
3.      Update set V of voting descriptors in D w.r.t. T_k and σ
4.      Partition T_k in set P_k of sub-trajectories w.r.t. V_k
    // SaCO phase (Sampling, Clustering, and Outlier detection)
5.  Find sampling set S consisting of the M most representative sub-trajectories
6.  Using set S and threshold ε, partition P = ∪P_k in a set C of M clusters and a set Out of outliers
7.  return (S, C, Out)
```

It is important to note that the number M of representatives (hence, the number of clusters) is not user-defined; rather, it is the algorithm that estimates it (in Line 6). As for parameters σ and ε that appear in Algorithm 1 (Line 3 and Line 7, respectively), σ controls how fast the voting influence decreases with distance, whereas ε acts as a lower bound threshold of similarity between representative and non-representative sub-trajectories, thus deciding whether a (non-representative) sub-trajectory will be flagged as outlier or not. These parameters will be explained in detail in the subsections that follow.

4.1 NaTS: Neighborhood-aware Trajectory Segmentation

We extend the concept of density-biased sampling (DBS), which was originally proposed for point datasets [18], to be applied to trajectory segments. According to DBS, the local density for each point of a set is approximated by the number of points in a surrounding region, divided by the volume of the region. In our case, adopting a voting process of trajectories in MOD as defined in [28], we define the representativeness of a 3D trajectory segment $e_{k,i}$ of a given trajectory T_k to be the number of 'votes' this segment collects from other trajectories w.r.t. their mutual distance. The overall voting collected by a segment (a value ranging from 0 to N) has the physical meaning of the number of other trajectories that co-exist with the trajectory that segment belongs to, both spatially and temporally. Intuitively, the voting results can be post-processed in order for us to be able to identify homogeneous (w.r.t. representativeness) sub-trajectories.

Formally, let V_k be the voting trajectory descriptor along the line segments of T_k, consisting of a series of L_k–1 components. Each component $V_{k,i}$ of this vector corresponds to the number of votes ("representativeness" value) that segment $e_{k,i}$, $i \in \{1, \ldots, L_k-1\}$, collected by the segments of the other trajectories. This representativeness value is based on a distance function $d(e_{k,i}, e_j)$ between two line segments $e_{k,i}$ and e_j, $k \neq j$. This distance function is defined as the definite integral of the time-varying distance $D_j(t)$ between the two segments during their common lifespan $[t_{j,start}, t_{j,end})$, following the approach proposed in [8]:

$$d(e_{k,i}, e_j) = \int_{t_{j,start}}^{t_{j,end}} D_j(t)\, dt \qquad (2)$$

As D_j follows a trinomial, this integral is efficiently approximated by the Trapezoid Rule:

$$\left(D_j(t_{j,start}) + D_j(t_{j,end})\right) \cdot (t_{j,start} - t_{j,end})/2$$

and can be computed in $O(1)$, as it has been already proved in [8].

Given the above distance function, the representativeness value is provided by the following voting function.

$$V(e_{k,i}, e_j) = e^{-\frac{d^2(e_{k,i}, e_j)}{2 \cdot \sigma^2}} \qquad (3)$$

As already mentioned, parameter $\sigma > 0$ controls the "voting influence", i.e. how fast $V(e_{k,i}, e_j)$ decreases with distance. It also holds that $V(e_{k,i}, e_j)$ is bounded in [0, 1]: it gets value 1 when the distance of the two segments is zero (i.e. the segments are identical) while very high distance results in voting value close to zero.

After the voting process takes place, the trajectory segmentation process gets into action. The goal of this step is to partition each trajectory into *homogenous representativeness* sub-trajectories, irrespectively of their shape complexity (recall the discussion about the snake-like trajectories in Figure 1). In order to perform neighbourhood-aware trajectory segmentation, we adopt the *Trajectory Segmentation Algorithm* (TSA), proposed in [28]. In other words, the result of the voting process is given as input to TSA, which provides as output the sub-trajectories along with their voting descriptors. More technically, let $P_{k,i}$, $i \in \{1, \ldots, LP_k\}$, be the i-th sub-trajectory of T_k, where LP_k denotes the number of partitions of T_k. Then, $VP_{k,i}$ is the voting descriptor formed by the representativeness values of the segments that belong to $P_{k,i}$. In other words, $VP_{k,i}$ shows how many trajectories find themselves to be similar to $P_{k,i}$. The interested reader is referred to [28] for the technical details of TSA.

Back to the example of Figure 1, the NaTS phase results in segmenting trajectory T_1 into three sub-trajectories (coloured red, blue, and black, respectively, in Figure 1(b)); similar for the other trajectories of the dataset. Thus, the overall result of this phase consists of 12 sub-trajectories along with their voting descriptors.

4.2 SaCO: Sampling, Clustering, and Outlier detection

As already mentioned, trajectory segmentation aims to provide homogeneous sub-trajectories according to their representativeness, i.e. with respect to their local similarity with other trajectories. On the other hand, the goal of sub-trajectory clustering is to partition the dataset into groups (clusters) of similar sub-trajectories. Therefore, in our proposal, we first select the appropriate sampling set S and then tackle the problem of clustering according to the following idea (quite popular, also in traditional data clustering): *each sub-trajectory in the sampling set is considered to be a representative around which a cluster will be formed*. So, our goal is that the sampling set should contain highly voted trajectories of the MOD which, at the same time, would cover the 3D space occupied by the entire dataset as much as possible in order for Eq. (1) to be maximized.

In order to achieve this goal, we propose the sampling to be done by maximizing a formula (see Eq. (4)) that would take into account the votes $VP_{k,i}$ collected by each sub-trajectory. Formally, let S denote the sampling set, so that $S_{k,i}$ is one, if sub-trajectory $P_{k,i}$ belongs to the sampling set, and zero otherwise. According to the previous discussion, the number of sub-trajectories that are represented in the sampling set S, should be maximized. This is formalized in Eqs. (4)-(6).

$$SR(S) = \sum_{k=1}^{N} \sum_{i=1}^{LP_k} S_{k,i} \cdot SR_{gain}(k,i) \quad (4)$$

where

$$SR_{gain}(k,i) = \sum_{j=1}^{|P_{k,i}|} VP^{P}_{k,i,j} \cdot Nl_{k,i,j} \cdot (1 - VP^{S}_{k,i,j}) \quad (5)$$

$$Nl_{k,i,j} = lifespan(e_{k,i,j}) / lifespan(T_k) \quad (6)$$

More precisely, $SR_{gain}(k,i)$ expresses the gain in $SR(S)$ if we add $P_{k,i}$ in S, $|P_{k,i}|$ denotes the number of line segments of $P_{k,i}$, $VP^{P}_{k,i,j}$ and $VP^{S}_{k,i,j}$ denote the votes in P and the votes in S, respectively, of the j-th line segment of $P_{k,i}$ and are calculated according to Eq. (3). As for $Nl_{k,i}$, it denotes the normalized lifespan descriptor of sub-trajectory $P_{k,i}$ w.r.t. lifespan of T_k, namely $Nl_{k,i,j}$ is the fraction of the duration of the j-th line segment of $P_{k,i}$ with respect to whole lifespan of T_k.

For this purpose, we follow the ideas included in the *Sub-trajectory Sampling Algorithm* (SSA), proposed in [28]. However, SSA is not appropriate for an efficient in-DBMS solution, which is one of our main objectives. Thus, we keep the main characteristics of the algorithm and adapt it in order to meet our specifications (described in detail in Section 5.2). In principle, the input of sampling algorithm is the set P of all sub-trajectories P_k, the set voting $VP_{k,i}$ and the normalized lifespan $Nl_{k,i}$ vectors of these sub-trajectories, all provided by the NaTS phase. The output of the sampling step is the sub-trajectory sampling set S consisting of M samples. Back to the example of Figure 1, this step results in selecting two sub-trajectories (samples), one out of the three red and one out of the four blue sub-trajectories.

As already mentioned, the population M of the samples is not user-defined; in contrary, it is dynamically estimated by SSA algorithm. As such, it provides a deterministic solution, in contrast to other probabilistic [18][27] or user-supervised, explorative sampling techniques [2].

What follows is the clustering step, which takes into account the sampling set S and the vector of votes (i.e. representativeness) $V(P_{k,i}, R_j)$ between, on the one hand, the non-representative $P_{k,i} \in P-S$ and, on the other hand, the representative sub-trajectories $R_j \in S$. Technically, $V(P_{k,i}, R_j)$ consists of $|P_{k,i}|$ elements, where each element represents the voting that takes place between the segments of $P_{k,i}$ and R_j. As illustrated in Eq. (1), we use the mean value $\overline{V(P_{k,i}, R_j)}$ of the vector values $V(P_{k,i}, R_j)$. Each of those values is computed by measuring the distance of the corresponding segment of $P_{k,i}$ from its nearest to R_j and then by applying the voting function of Eq. (3). Thus, it holds that $0 \leq \overline{V(P_{k,i}, R_j)} \leq 1$.

Concluding the discussion about Algorithm 1, in order to find the clusters that maximize Eq. (1), the sub-trajectories that are assigned to cluster $C(R_j)$ represented by sub-trajectory $R_j \in S$, are the ones that fulfil the following property:

$$C(R_j) = \left\{ P_{k,i} \in P - S : \overline{V(P_{k,i}, R_j)} \geq \overline{V(P_{k,i}, R_v)} \; \forall R_v \in S \wedge \overline{V(P_{k,i}, R_j)} \geq \varepsilon \right\} \quad (7)$$

and

$$C = \cup C(R_j) \quad (8)$$

On the other hand, the sub-trajectories that are considered outliers (thus forming the outliers set *Out*) are those failing to be assigned to a cluster, formally:

$$Out = P - C \quad (9)$$

As already discussed, parameter ε controls how far from a representative a non-representative should be positioned in order for the latter to be flagged as outlier. Back to the example of Figure 1, the clustering process presented above results in two clusters, formed around the red and the blue, respectively, representative sub-trajectory found in the sampling step. As a side effect, the black sub-trajectories are left out of the two clusters, thus they are flagged as outliers.

5. S^{2}T-CLUSTERING IN-DBMS

In this section, we present our methodology for the efficient in-DBMS development of S^{2}T-Clustering algorithm proposed in Section 4.

5.1 NaTS in-DBMS

NaTS phase of S^{2}T-Clustering algorithm (Lines 2–4 in Algorithm 1) consists of two steps: (a) voting among trajectory segments and (b) trajectory segmentation based on the resulted voting descriptors. An efficient in-DBMS solution should focus on the voting step (Lines 2–3), since TSA [28] that implements the segmentation step (Line 4) poses no special challenges; it is an efficient in-memory algorithm applied only on the voting descriptor of a single trajectory.

Back to the voting step, to meet its requirement we need an algorithm that takes as input a dataset $D = \{T_1, T_2, \ldots, T_N\}$ of trajectories, a trajectory $T_k \in D$ and $\sigma > 0$ parameter, and provides as output a voting descriptor (vector) V_k consisting of L_k–1 components, each corresponding to segment $e_{k,i}$, $i \in \{1, ..., L_k–1\}$, of trajectory T_k. For efficiency purposes, [28] implemented the demanding voting process by using an incremental nearest neighbour (INN) algorithm. However, given the specifications posed in the introduction of this paper, INN is not a choice due to the fact that the access methods supported by real ORDBMS (e.g. the GiST interface in PostgreSQL) do not support the incremental paradigm. This implies that, in our case, we are directed to queries natively supported by ORDBMS, such as typical range and NN queries.

Let us now discuss the design and implementation options we have in-DBMS. Dataset D corresponds to a relation with tuples in the form <*t_id*, *s_id*, $e_{k,i}$>, where *t_id* (*s_id*) is the trajectory (segment, respectively) identifier and $e_{k,i}$ corresponds to the 3D segment, upon which a 3D-R-tree index is built. Nevertheless,

this setting is straight-forwardly realized in the well-known PostGIS spatial extension of PostgreSQL using 3D GiST. (Note, however, that PostGIS handles time- dimension as simply as a (third) z- spatial dimension, next to x- and y- dimensions.) An important issue has also to do with the realization of Eq. (3) that provides the voting between two segments: theoretically, a segment may vote (though close to zero) even if it is found very far from the target segment. However, this is not realistic in DBMS implementations. As such, we introduce *s_buffer*, a spatial threshold for distance between two segments, above which there is no need to calculate this distance. In the case where the application user has limited knowledge about space-time properties of the dataset, this parameter can be tuned to be the maximum value resulting in a very low (close to zero) voting as computed by Eq. (3). This is achieved as follows: by reversing Eq. (3), we obtain Eq. (10) that defines an upper bound for *s_buffer*.

$$d \leq \sqrt{-2\sigma^2 \cdot \ln(\varepsilon)} \qquad (10)$$

Thus, d values higher than the upper bound set in Eq. (10) are not expected to contribute to the quality of the clustering.

Given the above setting, voting can be implemented using at least two alternatives, called Baseline-I and Baseline-II, respectively. Baseline-I solution performs $\sum_k (L_k - 1)$ range queries in the 3D-R-tree, where each query window corresponds to the MBB of a segment, enlarged by *s_buffer*; hence, the total number of range queries equals to the total number of segments in D, a fact that turns this solution to be expensive in disk accesses. On the other hand, Baseline-II solution performs N range queries in the 3D-R-tree, where each query window corresponds to the MBB of a trajectory, again enlarged by *s_buffer*; hence, the total number of range queries equals to the number of trajectories in D. Obviously, the second solution is much cheaper in disk accesses regarding the index but, unfortunately, imposes a heavy refinement step because of the volume of the trajectory MBB. Anyway, both approaches need a refinement step to calculate voting descriptor $V_{k,i}$, which involves distance calculations.

In the following paragraphs, we present an alternative (third) approach for addressing the voting step, which is the most demanding step in S^{2}T-Clustering algorithm and, as such, it needs special care. In particular, we follow a filter-and-refinement approach that utilizes a range-like query, called *Trajectory Buffer Query* (TBQ). TBQ takes as input a trajectory, enlarges it by *s_buffer*, and returns the segments that overlap with the sequence of the enlarged MBBs of the trajectory's segments. The TBQ rationale is to efficiently retrieve those segments in D that are "around" a given trajectory, where "around" is defined by *s_buffer*. Figure 2 illustrates the Trajectory Buffer TB_k of a trajectory T_k.

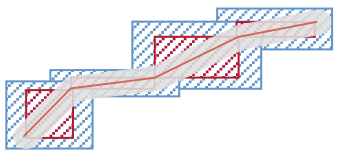


Figure 2. The Trajectory Buffer TB_k (i.e. the sequence of the blue MBBs) of a trajectory T_k.

It is obvious that our proposal follows a trajectory-based approach (i.e. similar to the Baseline-II technique), but for each trajectory it minimizes the filtering step by diminishing the dead space of the query, and thus minimizes the expensive refinement step. In turn, this implies changing the default search strategy of the 3D-R-tree over GiST that will reduce the time needed to compare a node entry with the trajectory buffer that is passed as predicate to the index. This is achieved by the *Consistent* method of the GiST extensibility interface [14], which contains the comparison logic between an index node entry of GiST and the trajectory buffer. Algorithm 2 outlines TBQ whereas Algorithm 3 presents the adapted *Consistent* method of the GiST interface.

Algorithm 2. Trajectory Buffer Query (TBQ)
Input: pg3D-R-tree *root*, trajectory T_k, parameter *s_buffer*
Output: set of segments that overlap with TB_k
1. $TB_k \leftarrow$ *TrajectoryBuffer*(T_k, *s_buffer*)
2. *root.depth-first-search*(*Consistent*, TB_k)

Algorithm 3. Consistent
Input: Trajectory Buffer TB_k, current index entry *E*
Output: Boolean
1. **if** *E* is in a leaf node **then**
2. **if** MBB(*E.segment*) overlaps MBB(TB_k) **then**
3. **for** each $MBB_i \in TB_k$ **do**
4. **if** *E.segment* overlaps MBB_i **then**
5. **return** true
6. **else** // E is in a non-leaf node
7. **if** *E.box* overlaps MBB(TB_k) **then**
8. **for** each $MBB_i \in TB_k$ **do**
9. **if** *E.box* overlaps MBB_i **then**
10. **return** true
11. **return** false

Recall that *Consistent* decides whether the depth-first search should visit a child of the current entry or not (if the entry belongs to a non-leaf node) or, in case the entry belongs to a leaf node, checks whether to return the segment pointed by the leaf entry. After this remark, the depth-first search driven by *Consistent* in Algorithm 3 is easy to be followed: *Consistent* returns true if the MBB of the entry overlaps with one of the MBBs forming the trajectory buffer TB_k (Lines 5 and 10, for leaf and non-leaf nodes, respectively). Before this check takes place, a brute filtering is applied by checking whether the MBB of the entry overlaps the entire MBB of TB_k (Lines 2 and 7, respectively).

5.2 SaCO in-DBMS

In this section, we discuss the in-DBMS development of SaCO, i.e. the second phase of S^{2}T-Clustering. SaCO phase (Lines 5–6 in Algorithm 1) also consists of two steps: (a) sampling of the most representative sub-trajectories (Line 5) and (b) clustering around samples and outlier detection (Line 6).

Regarding the sampling step, we adopt the SSA algorithm [28] as a starting point and we improve it with two crucial modifications, focusing on the efficiency and the quality, respectively, of the samples selected. The first improvement is that the voting method that is inherent in the sampling process follows the much more efficient approach presented earlier rather than the one presented in [28]. The second modification is about the selection of an even better set of representatives; as proposed in [28], SSA selects representatives as long as (a) the top-k number of representatives is less than a user-defined threshold (i.e. parameter M that acts as an upper bound for the selected representatives) and (b) the optimization criterion is satisfied (see Eq. (4) and (5)). In fact, SSA selects the highly voted sub-trajectories, while at the same time it tries to penalize sub-trajectories that are very close to already selected representatives. Sometimes this automatic penalization fails, resulting to very similar representatives. In contrast, in our case, as the representatives are employed as cluster pivots, when a new representative is selected, it is further examined whether it is similar with one of the already selected representatives. In such a case, it is not selected and the algorithm evaluates the next candidate sub-trajectory. The similarity criterion is the same with the one adopted for the clustering, i.e. Eq. (7).

What follows is the final step, that of clustering and outlier detection. For this purpose, we follow an index-based, greedy approach that takes advantage of the TBQ query, which is applied on the results of the SSA algorithm, so as to form clusters around the sampled sub-trajectories. To this end, we propose the so-called *Sub-trajectory Clustering Algorithm* (SCA). SCA, listed in Algorithm 4, receives as input set P of sub-trajectories, set S of representatives, as it was produced by the (modified) SSA, and threshold parameter ε. The output of the method is the final result of S^{2}T-Clustering, i.e. sets C and *Out*, with the clusters and outliers, respectively.

Algorithm 4. SCA

Input: set P of sub-trajectories, set S of representatives, parameter ε
Output: set C of clusters, set *Out* of outliers

```
1.  Out = P − S
2.  for each R_j ∈ S do
3.      C_j ← {R_j}
4.  for each R_j ∈ S do
5.      TBQ_j ← TBQ(Out, R_j, s_buffer)
6.      for each e_j,f ∈ R_j do
7.          TBQ_j,f ← overlaps(TBQ_j, extend(e_j,f, s_buffer))
8.      for each P_k,i in {TBQ_j,f}, f ∈ [1, |R_j|] do
9.          v ← V(P_k,i , R_j)
10.         if v > ε and v > old_v_k,i then
11.             C_j ← C_j ∪ {P_k,i}
12.             flag P_k,i as clustered in Out
13.             old_v_k,i ← v
14. for each P_k,i in Out do
15.     if P_k,i is flagged as clustered then
16.         Out ← Out − {P_k,i};
17. return (C, Out)
```

Initially, the sub-trajectories are organized in two sets (implemented as relations in DBMS), one containing the sampling set sorted by the order of their selection and the other containing the remaining data, while each cluster is initialized by a representative sub-trajectory from the sampling set. As such, each representative sub-trajectory constitutes the first member (seed) of the corresponding cluster (Lines 1-3). Then, we apply a two-step filtering procedure so as to increase the efficiency of the algorithm. At the first step, for each cluster seed R_j, we apply a TBQ query, which returns the segments that are "close" to the cluster seed (Line 5). Subsequently, for each segment $e_{j,f}$ belonging to the specific representative R_j, we apply a spatiotemporal range query with the same spatial component as that of the TBQ query (Line 7). This spatiotemporal range query is performed in order to identify the segments that are "close enough" to $e_{j,f}$ and, hence, qualify to proceed to the voting procedure w.r.t. R_j. Subsequently, for each non-clustered $P_{k,i}$, we calculate the average voting that R_j receives (Line 9). By taking into account parameter ε discussed earlier, we assign it to cluster C_j mastered by R_j (Line 11) and mark it as clustered (Line 12). Through this process, in the case where $P_{k,i}$ belongs to the result of more than one TBQ searches, it is assigned to the representative that has achieved the highest voting.

6. EXPERIMENTAL STUDY

In this section, we present the results of our experimental study. All experiments were conducted on an Intel Xeon X5675 Processor 3.06GHz with 48GB memory, running on Debian Release 7.0 (wheezy) 64-bit. The proposed algorithms were implemented on top of a PostgreSQL 9.4 server with the default configuration for its memory parameters. We should clarify that in our implementation, which exploits on the extensibility interface given by PostgreSQL, we have defined and implemented from scratch datatypes and operands conforming to the whole discussion so far, resulting in the so-called Hermes@PostgreSQL [16], which is completely independent from PostGIS. This implies that the 3D-R-tree has also been implemented from scratch (on top of GiST); we call it pg3D-R-tree (see the input of TBQ in Algorithm 2).

A notable difference of our pg3D-R-tree from the PostGIS implementation of the 3D-R-tree is that, in our case, the entries of the leaf nodes are 3D segments rather than 3D boxes. This is an implicit assumption in the *Consistent* algorithm (see e.g. Line 2 in Algorithm 3), which allows us to avoid additional I/O operations. The outline of our experimental study is as follows: First, we study the robustness of S^{2}T-Clustering by using a synthetic dataset (where we know the ground truth) in order to (a) evaluate the sensitivity of our proposal w.r.t. various parameters and (b) validate whether our approach succeeds to discover the underlying clusters (and outliers). Then, a set of experiments is performed in order to evaluate the efficiency and scalability of S^{2}T-Clustering. These experiments are performed using three different approaches: the two baseline solutions and our solution based on TBQ, as they were presented in Section 5.

6.1 Datasets

The three datasets we used in our experimental study, one synthetic (SMOD) and two real datasets (IMIS, GeoLife), are presented in the following paragraphs.

SMOD - Synthetic MOD (SMOD)[1] consists of 400 trajectories and is used for the ground truth verification (see the discussion about ground truth below). The creation scenario of the synthetic dataset is the following: the objects move upon a simple graph that consists of the following destination nodes (points) with coordinates: A(0,0), B(1,0), C(4,0) and D(2,1). Half of the objects move with normal speed (2 units per second) and another half move with high speed (5 units per second). Figure 3 illustrates the 2D map of the SMOD consisting of three one-directional (A → B, B → D, D → C) and one bi-directional road (B ⇆ C). All objects move under the following scenario, for a lifetime of 100 seconds:

- (normal movement – 99% of the trajectories) All objects start from point A towards point B; the high-speed objects start at t = 0 sec and the normal-speed objects start at t = 20 sec. When an object arrives at B, it ends its trajectory with a probability of 15%; otherwise, it continues with the same speed to the next point. If there exist more than one option for the next point, it decides randomly about the next destination.
- (abnormal movement – 1% of the trajectories) A few outlier objects follow a random movement in space (other than these roads) with a speed that is updated randomly.

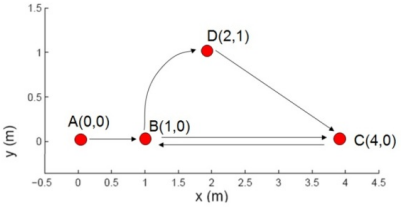


Figure 3. The 2-D map of SMOD.

The ground truth of the clusters that are hidden in SMOD can be inferred by the description of the dataset itself. In particular, eight clusters of sub-trajectories (as well as a set of outliers) are identified. Table 2 lists the eight clusters along with their spatial (2nd column) and temporal projection (3rd column).

[1] Publicly available at chorochronos.datastories.org repository under the name 'smod'.

Table 2. The ground truth hidden in SMOD

Cluster	Path	Time periods (clusters)
#1, #2	A→B	[0, 0.2], [0.2, 0.7]
#3, #4	B→C	[0.2, 0.8], [0.7, 1.2]
#5, #6	B→D	[0.2, 0.52], [0.7, 1.2]
#7	C→B	[0.8, 1]
#8	D→C	[0.52, 1]

As for real datasets, **GeoLife** [47] consists of the trajectories of 178 users in a period of more than four years; this dataset represents a wide range of movements, including not only urban transportation (e.g. from home to work and back) but also different kinds of activities, such as sports activities, shopping, etc. Finally, **IMIS**[2] is a real AIS dataset consisting of the trajectories of 637 ships moving in the Greek seas for one week. Table 3 presents the statistics of the three datasets.

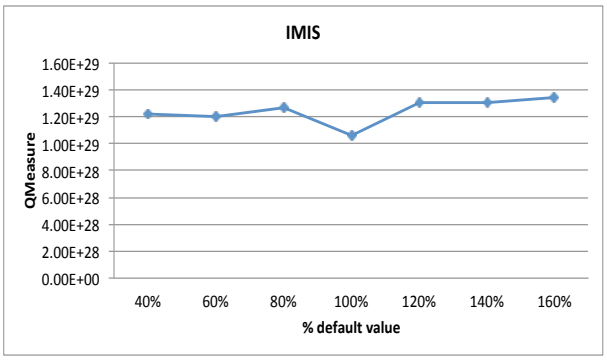
6.2 Quality of Clustering Analysis

In this section, we perform a sensitivity analysis in order to explore the effect on the quality of clustering when setting different values on certain parameters. The quality of the clustering is calculated through two different measures: QMeasure [21] and SRD (see Eq. (1)). We should mention that the lower the QMeasure the higher the quality; on the other hand, the higher the SRD the higher the quality. Regarding parameter settings, as our approach shares similar concepts with the sampling methodology of [28], we followed the best practices presented in that work. More specifically, parameter σ was set to 0.1% of the dataset diameter while ε was set to 10^{-3}. Regarding *s_buffer*, it was automatically set according to Eq. (10) as default value and we experimented with values around the default.

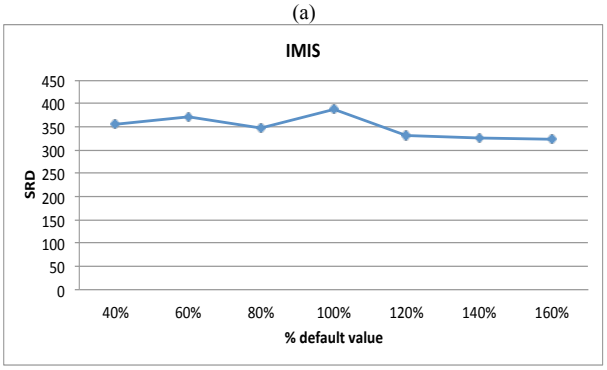
Table 3. Dataset Statistics

Statistic	SMOD	GeoLife	IMIS
# Trajectories	400	18,668	5110
# Segments	35,273	24,159,325	443,657
Dataset Duration (hh:mm:ss)	0:02:00	1932 days 22:59:48	6 days 19:59:53
Avg. Sampling Rate (hh:mm:ss)	0:00:01	0:00:08	0:18:02
Avg. Segment Length (m)	8	72	1545
Avg. Segment Speed (m/s)	7.83	5.01	7.03
Avg. Trajectory Speed (m/s)	2.86	3.91	4.52
Avg. # Points per Trajectory	89	1295	88
Avg. Trajectory Duration (hh:mm:ss)	0:01:28	2:43:15	11:33:45
Avg. Trajectory Length (m)	691	93,046	134,148

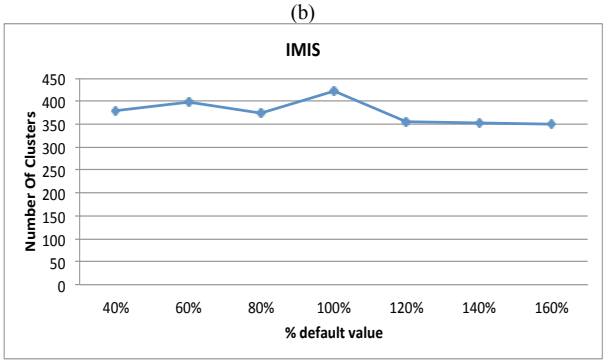
[2] Publicly available at chorochronos.datastories.org under the name 'imis1week'.



(a)



(b)



(c)

Figure 4. The effect on (a) QMeasure, (b) SRD, (c) the discovered number of clusters, when varying *s_buffer* parameter around its default value.

The first set of experiments is about the sensitivity of S^2T-Clustering w.r.t. *s_buffer*. Figure 4 illustrates the results over the IMIS dataset. In particular, we used the default value (labelled 100% in the x-axis of the charts) as well as 6 values around it (labelled 40%, 60%, 80%, 120%, 140%, 160%). As one can easily observe, the quality of the clustering, measured either by QMeasure or SRD, remains more or less stable and follows the trend of the number of clusters identified. Moreover, in both QMeasure and SRD, the best quality appears when *s_buffer* is set to its default value (*d*).

We repeated the same experiment over GeoLife and resulted in similar conclusions. Considering the above analysis, the value for *s_buffer* used in the remainder of our experimental study is the default value provided by Eq. (10).

In a second set of experiments, we applied our proposal to the SMOD dataset, which is ideal for the purposes of testing the quality of our algorithm. In order to measure the stability of our method to noise effects, we have added Gaussian white noise of different Signal to Noise Ratio (SNR) levels, measured in db, to the spatial coordinates of SMOD. All the subsequent experiments have been repeated with SNR = 30db and SNR = 50db and the results were the same. Therefore, we present only the case with the SNR =30db.

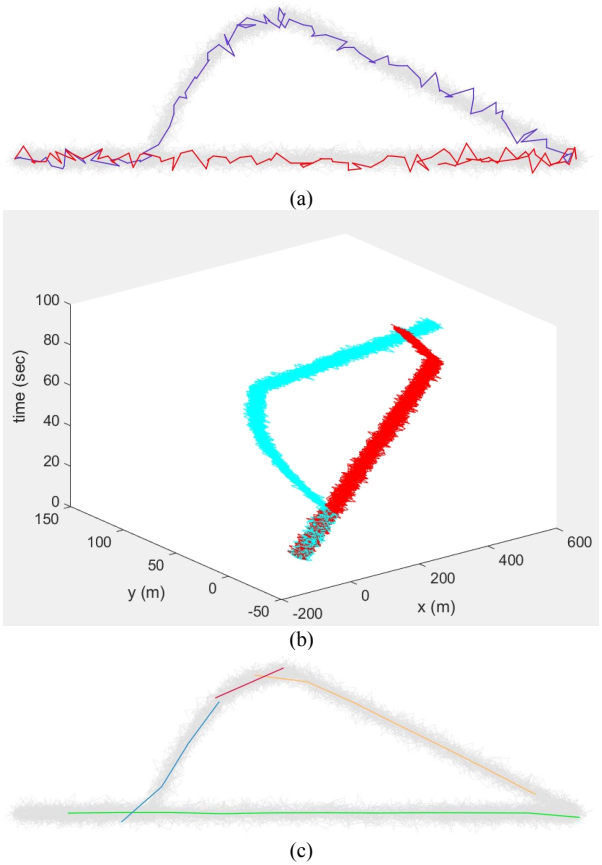


Figure 5. Visualization of the clusters' representatives provided by: S^{2}T-Clustering in (a) 2D and (b) 3D, (c) TRACLUS, when applied to a subset of SMOD consisting of 2 patterns.

First, we applied both S^{2}T-Clustering and TRACLUS [21] over a subset of SMOD that consists only of the trajectories that move throughout the whole lifespan of the dataset, thus limiting the ground truth to two clusters. In Figure 5(a) and Figure 5(c) we visualize only the representatives of each cluster, while in Figure 5(b) we provide a 3D illustration of the data used in the case of Figure 5(a). Note that S^{2}T-Clustering discovers the two clusters, while TRACLUS discovers several linear patterns; see Figure 5(a) vs. Figure 5(c).

Subsequently, we applied both S^{2}T-Clustering and TRACLUS to the entire SMOD, for which we have knowledge of the ground truth. In Figure 6(a) and Figure 6(c), we present the results of the S^{2}T-Clustering and TRACLUS, respectively. Moreover, in order to better comprehend the temporal dynamics of the dataset we provide a 3D illustration in Figure 6(b). According to this experiment, S^{2}T-Clustering effectively discovers all eight clusters (as well as the noisy sub-trajectories, depicted in black color in Figure 6(b)), thus S^{2}T-Clustering is not affected by the trajectories' shape, yielding an effective and robust approach for the discovery of linear and non-linear patterns. On the contrary, TRACLUS fails to identify the hidden ground truth in this SMOD due to the fact that it ignores the time dimension. Interestingly, TRACLUS discovers almost the same sets of representatives when applied to either a subset of or the entire SMOD; see Figure 5(c) vs. Figure 6(c).

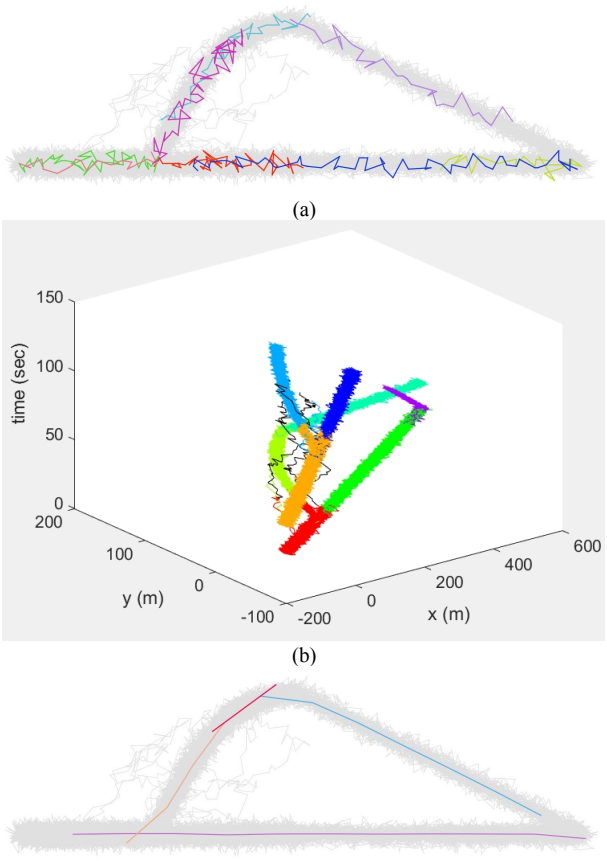


Figure 6. Visualization of the clusters' representatives provided by: (a) S^{2}T-Clustering in (a) 2D and (b) 3D, (c) TRACLUS, when applied to the entire SMOD consisting of 8 patterns.

In order to evaluate the accuracy of our proposal in a quantified way, we further employed F-Measure in SMOD. In detail, we built 8 datasets, with the first consisting of the sub-trajectories of the first cluster only, the second consisting of the sub-trajectories of the first and the second cluster only, and so on, until the eighth dataset, which consisted of the sub-trajectories of all eight clusters; all eight datasets appeared in two variations: including or not the set of outliers. For each dataset, we applied S^{2}T-Clustering and calculated F-Measure; Figure 7 illustrates this quality criterion by increasing the number of clusters. It is evident that S^{2}T-Clustering turns out to be very robust, achieving always precision and recall values over 92.3%, while the outliers are always detected correctly.

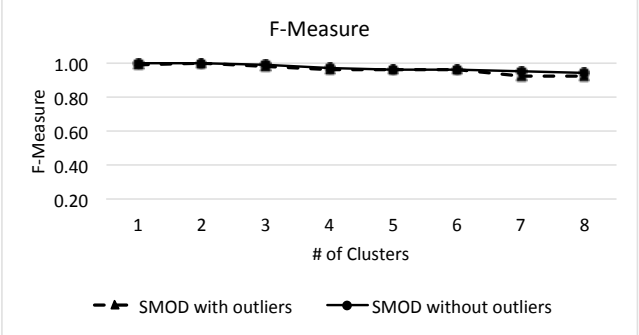


Figure 7. Quality of S^{2}T-Clustering w.r.t. number of clusters.

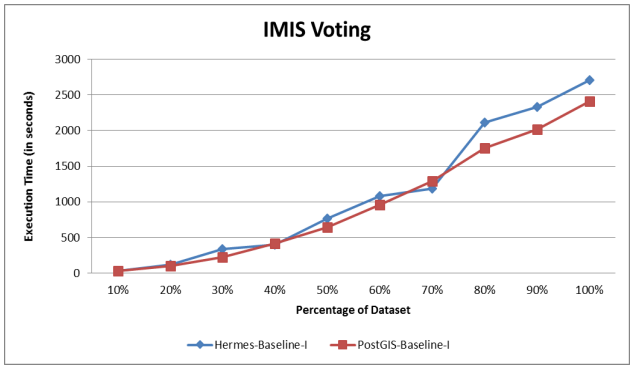
6.3 Efficiency and Scalability

In order to study the efficiency and scalability of our proposal we followed two competing approaches: Hermes@PostgreSQL [16], implemented according to the discussion in Section 5, vs. PostGIS extension of PostgreSQL that simulated the two baseline solutions presented in Section 5.1.

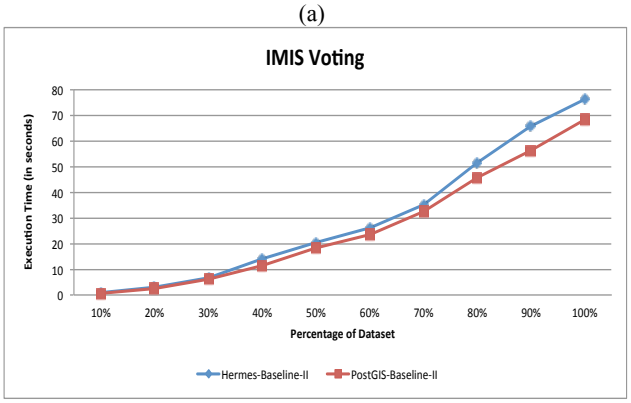
We have noticed that the implementation of the 3D-R-tree in PostGIS suffers from rounding errors because it uses 32-bit IEEE floating-point numbers to store the coordinates [35]. In our experiments we observed that the MBB of a trajectory or a segment was always enlarged due to this rounding, thus making the overlap query in PostGIS return more segments than our implementation. Since this made the comparison between the two systems unfair, we simulated PostGIS inside Hermes, in other words, also the baseline solutions were simulated inside Hermes (thus, making all solutions run under the same framework).

In the charts that follow, we denote the implementation of Baseline-I and Baseline-II solutions implemented both in Hermes and in PostGIS as {Hermes | PostGIS}-Baseline-{I | II}, i.e. four different implementations.

In particular, Figure 8 illustrates the execution time of the voting step for the IMIS dataset when varying the dataset size (i.e. the number of trajectories). Obviously, the two implementations present similar performance, with the PostGIS implementation performing slightly better mainly due to the fact that the size of index node entries in PostGIS (which uses 32-bit numbers for storing the temporal dimension) is slightly less than that of Hermes (which uses 64-bit numbers).



(a)



(b)

Figure 8. Comparing the performance of baseline solutions: (a) Baseline-I; (b) Baseline-II.

We repeated the same experiment with the GeoLife dataset and the results lead to similar conclusions, thus they are excluded due to space limitations.

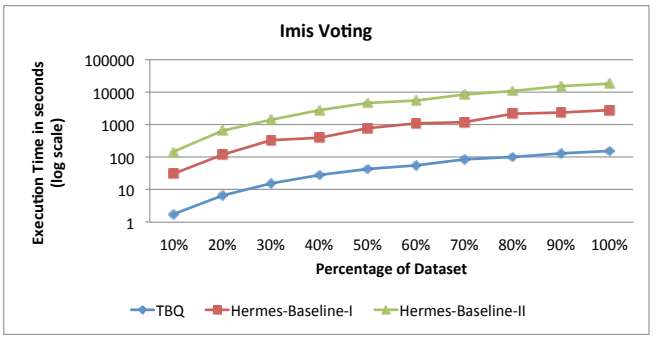
Based on the above results, in the remainder of the experimental study, the scalability study is conducted using the Hermes implementation of the algorithms. As illustrated in Figure 9(b), all three approaches (Baseline-I, Baseline-II and TBQ, presented in Section 5.1) perform similarly on the IMIS dataset as far as it concerns the segmentation, sampling and clustering steps of the algorithm (please note that y-axis is at log scale). The crucial difference is at the expensive voting step, where TBQ significantly outperforms the two baseline solutions by almost two orders of magnitude; this is illustrated in Figure 9(a) whereas in Figure 9(c) we present the accumulated processing time.

Due to the fact that the overall performance is dominated by the performance of the voting step, we further studied this step over the GeoLife dataset. As it can be observed in Figure 9(d), the behavior of the voting step of S^2T-Clustering over GeoLife is slightly different from that over IMIS. TBQ still outperforms both Baseline-I and Baseline-II solutions by several orders of magnitude, but in the case of GeoLife, Baseline-II outperforms Baseline-I. This can be explained by the fact that GeoLife consists of trajectories with significantly larger number of segments than IMIS (recall the statistics in Table 3). This fact leads Baseline-I to perform considerably more lookups in the index.

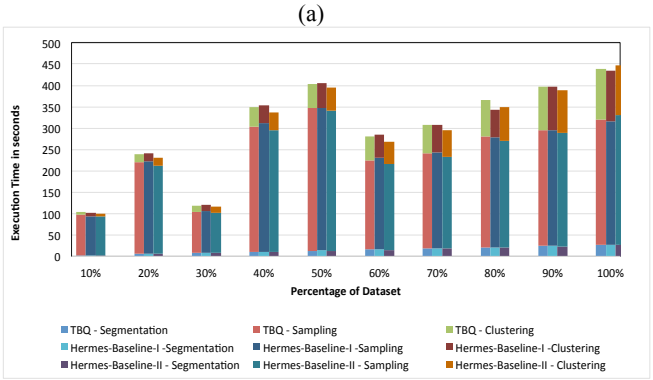
7. CONCLUSIONS

In this paper, we discussed the problem of sub-trajectory clustering and outlier detection in trajectory databases, aiming to take both space and time information into consideration. In particular, we proposed S^2T-Clustering that is novel not only because it solves the problem more effectively than the state-of-the-art (namely, TRACLUS), but also for an additional, quite important reason: our proposal is designed in-DBMS, i.e., it performs as a query operator in a real MOD engine over an extensible DBMS (namely, PostgreSQL in our current implementation). Having such functionality in their hands, data scientists are able to perform cluster analysis via simple SQL in real DBMS, where concurrency and recovery issues are taken into consideration. Moreover, our algorithm is boosted by an efficient index-based Trajectory Buffer Query (TBQ) that speeds up the overall process, resulting in a scalable solution, outperforming the state-of-the-art in-DBMS solutions supported by PostGIS by several orders of magnitude.

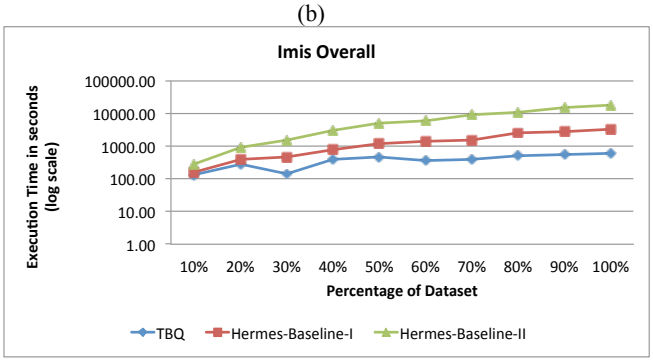
As a next step, inspired by the research agenda of the big data era, we plan to investigate real-time and incremental solutions, exploiting on modern in-memory DBMS architectures.



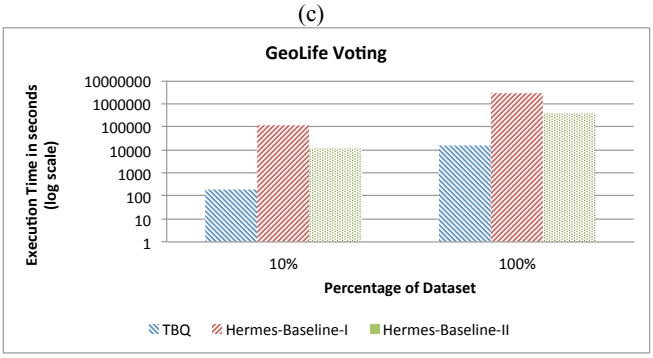
(a)



(b)



(c)



(d)

Figure 9. Step-by-step execution time of S^{2}T-Clustering: (a) voting over IMIS; (b) segmentation/sampling/clustering over IMIS; (c) overall over IMIS; (d) voting over GeoLife.

8. ACKNOWLEDGMENTS

This work was partially supported by project datACRON, which has received funding from the European Union's Horizon 2020 research and innovation Programme under grant agreement No 687591.

9. REFERENCES

[1] Almeida, V.T., Güting, R.H., & Behr, T. 2006. Querying moving objects in secondo. In Proceedings of MDM.

[2] Andrienko, G., Andrienko, N., Rinzivillo, S., Nanni, M., and Pedreschi D. 2009. A visual analytics toolkit for cluster-based classification of mobility data. In Proceedings of SSTD.

[3] Ankerst, M., Breunig, M. M., Kriegel, H.-P. and Sander, J. 1999. Optics: Ordering points to identify the clustering structure. In Proceedings of SIGMOD.

[4] Benkert, M., Gudmundsson, J., Hubner, F. and Wolle T. 2006. Reporting flock patterns. In Proceedings of ESA.

[5] Cadez, I. V., Gaffney, S., and Smyth, P. 2000. A general probabilistic framework for clustering individuals and objects. In Proceedings of KDD.

[6] Dodge, S., Weibel, R., and Lautenschütz, A.-K. 2008. Towards a taxonomy of movement patterns. Journal of Information Visualization. 7(3), 240-252.

[7] Ester, M., Kriegel, H.-P., Sander, J., Xu, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of KDD.

[8] Ferreira, N., Klosowski, J.T., Scheidegger, C.E., Silva, C.T. 2013. Vector Field k-Means: Clustering Trajectories by Fitting Multiple Vector Fields. In Proceedings of EuroVis.

[9] Frentzos, E., Gratsias, K., and Theodoridis, Y. 2007. Index-based most similar trajectory search. In Proceedings of ICDE.

[10] Gaffney, S., and Smyth, P. 1999. Trajectory clustering with mixtures of regression models. In Proceedings of KDD.

[11] Giannotti, F. and Pedreschi, D. 2008. Mobility, Data Mining and Privacy, Geographic Knowledge Discovery. Springer.

[12] Giannotti, F., Nanni, M. Pedreschi, D. Pinelli, F., Renso, C., Rinzivillo, S. and Trasarti, R. 2011. Unveiling the complexity of human mobility by querying and mining massive trajectory data. The VLDB Journal, 20(5): 695-719.

[13] Hadjieleftheriou, M., Kollios, G., Gunopulos, D. and Tsotras, V.J., 2006. Indexing Spatio-Temporal Archives, VLDB J., vol. 15, no. 2, pages 143-164.

[14] Hellerstein, J., Naughton, J. and Pfeffer, A. 1995. Generalized Search Trees for Database Systems. In Proceedings of VLDB.

[15] Hung, C.-C., Peng, W.-C., Lee, W.-C. 2015. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. The VLDB Journal, 24(2):169-192.

[16] Hermes@PostgreSQL MOD engine. URL: http://infolab.cs.unipi.gr/hermes/

[17] Jeung, H., Yiu, M. L., Zhou, X., Jensen, C., and Shen, H. T. 2008. Discovery of convoys in trajectory databases. In Proceedings of VLDB.

[18] Kalnis, P., Mamoulis, N., Bakiras, S. 2005. On discovering moving clusters in spatio-temporal data. In Proceedings of SSTD.

[19] Kollios, G., Gunopulos, D., Koudas, N., and Berchtold, S. 2003. Efficient biased sampling for approximate clustering and outlier detection in large datasets. IEEE Transactions on Knowledge and Data Engineering, 15(5):1170-1187.

[20] Kornacker, M. 1999. High-Performance Extensible Indexing. In Proceedings of VLDB.

[21] Lee, J.-G., Han, J., and Whang, K.-Y. 2007. Trajectory clustering: a partition-and-group framework. In Proceedings of SIGMOD.

[22] Li, Y., Bailey, J. Kulik, L. 2015. Efficient mining of platoon patterns in trajectory databases. Data & Knowledge Engineering, 100(PA):167-187.

[23] Li, Z., Ding, B., Han, J., Kays, R. 2010. Swarm: Mining relaxed temporal moving object clusters. In Proceedings of the VLDB Endowment 3(1-2):723–734.

[24] Li, Z., Lee, J.G., Li, X., Han, J. 2010 Incremental clustering for trajectories, In Proceedings of DASFAA.

[25] Li, Z., Ji, M., Lee, J.-G., Tang, L.-A., Yu, Y., Han, J., Kays, R. 2010. MoveMine: mining moving object databases. In Proceedings of SIGMOD.

[26] Nanni, M., and Pedreschi, D. 2006. Time-focused clustering of trajectories of moving objects. Journal of Intelligent Information Systems, 27(3):267-289.

[27] Nanopoulos, A., Theodoridis, Y., and Manolopoulos, Y. 2006. Indexed-based density biased sampling for clustering applications. Data and Knowledge Engineering, 57(1):37-63.

[28] Panagiotakis, C., Pelekis, N., Kopanakis, I., Ramasso, E., and Theodoridis, Y. 2012. Segmentation and sampling of moving object trajectories based on representativeness. IEEE Transactions on Knowledge and Data Engineering, 24(7):1328-1343.

[29] Pelekis, N. and Theodoridis, Y. 2014. Mobility Data Management and Exploration. Springer.

[30] Pelekis, N., Andrienko, G., Andrienko, N., Kopanakis, I., Marketos, G., Theodoridis, Y. 2011. Visually Exploring Movement Data via Similarity-based Analysis. Journal of Intelligent Information Systems, 38(2):343-391.

[31] Pelekis, N., Frentzos, E., Giatrakos, N., and Theodoridis, Y. 2008. HERMES: Aggregative LBS via a trajectory DB engine. In Proceedings of SIGMOD.

[32] Pelekis, N., Kopanakis, I., Kotsifakos, E., Frentzos, E. and Theodoridis, Y. 2011. Clustering uncertain trajectories. Knowledge and Information Systems, 28(1):117-147.

[33] Pelekis, N., Panagiotakis, C., Kopanakis, I., and Theodoridis, Y. 2010. Unsupervised trajectory sampling. In Proceedings of ECML-PKDD.

[34] Pfoser, D., Jensen, C.S., and Theodoridis, Y. 2000. Novel approaches to the indexing of moving object trajectories. In Proceedings of VLDB.

[35] Ramsey, P. (on behalf of PostGIS), personal communication.

[36] Tang, L.A., Zheng, Y., Yuan, J., Han, J., Leung, A., Hung, C. and Peng, W. 2012. Discovery of Traveling Companions from Streaming Trajectories. In Proceedings of ICDE.

[37] Tang, L.A., Zheng, Y., Yuan, J., Han, J., Leung, A., Peng, W. and Porta, T. L. 2012. A Framework of Traveling Companion Discovery on Trajectory Data Streams. ACM Transactions on Intelligent Systems and Technology, 5(1).

[38] Theodoridis, Y., Vazirgiannis, M. and Sellis, T. 1996. Spatio-Temporal Indexing for Large Multimedia Applications. In Proceedings of ICMS.

[39] Wang, S., Wu, L., Zhou. F. Zheng, C., Wang, H. 2015. Group Pattern Mining Algorithm of Moving Objects' Uncertain Trajectories. International Journal of Computers, Communications & Control, 10(3):428-440.

[40] Wu, F., Lei, T.K.H., Li, Z. Han, J. 2014. MoveMine 2.0: mining object relationships from movement data. In Proceedings of VLDB.

[41] Xu, H., Zhou, Y., Lin, W., Zha, H. 2015. Unsupervised Trajectory Clustering via Adaptive Multi-Kernel-based Shrinkage. In Proceedings of ICCV.

[42] Yuan, G., Sun, P., Zhao, J., Li, D. Wang, C. 2016. A review of moving object trajectory clustering algorithms. Artificial Intelligence Review, 1-22.

[43] Zhang, T., Ramakrishnan, R., and Livny, M. 1996. Birch: An efficient data clustering method for very large databases. In Proceedings of SIGMOD.

[44] Zheng, K., Zheng, Y., Yuan, N. J., Shang, S. 2013. On Discovery of Gathering Patterns from Trajectories. In Proceedings of ICDE.

[45] Zheng, K., Zheng, Y., Yuan, N. J., Shang, S., Zhou., X. 2014. Online Discovery of Gathering Patterns over Trajectories. IEEE Transaction on Knowledge and Data Engineering, 26(8):1974-1988.

[46] Zheng, Y. 2015. Trajectory Data Mining: An Overview. ACM Transactions on Intelligent Systems and Technology, 6(3).

[47] Zheng, Y., Xie, X., Ma, W.-Y. 2010. GeoLife: A Collaborative Social Networking Service among User, location and trajectory. IEEE Data Engineering Bulletin, 33(2):32-40.