On Temporal-Constrained Sub-Trajectory Cluster Analysis

Nikos Pelekis[†] · Panagiotis Tampakis · Marios Vodas · Christos Doulkeridis · Yannis Theodoridis

Abstract Cluster analysis over Moving Object Databases (MODs) is a challenging research topic that has attracted the attention of the mobility data mining community. In this paper, we study the temporal-constrained sub-trajectory cluster analysis problem, where the aim is to discover clusters of sub-trajectories given an ad-hoc, user-specified temporal constraint within the dataset's lifetime. The problem is challenging because: (a) the time window is not known in advance, instead it is specified at query time, and (b) the MOD is continuously updated with new trajectories. Existing solutions first filter the trajectory database according to the temporal constraint, and then apply a clustering algorithm from scratch on the filtered data. However, this approach is extremely inefficient, when considering explorative data analysis where multiple clustering tasks need to be performed over different temporal subsets of the database, while the database is updated with new trajectories. To address this problem, we propose an incremental and scalable solution to the problem, which is built upon a novel indexing structure, called Representative Trajectory Tree (*ReTraTree*). ReTraTree acts as an effective spatio-temporal partitioning technique; partitions in *ReTraTree* correspond to groupings of sub-trajectories, which are incrementally maintained and assigned to representative (sub-)trajectories. Due to the proposed organization of sub-trajectories, the problem under study can be efficiently solved as simply as executing a query operator on *ReTraTree*, while insertion of new trajectories is supported. Our extensive experimental study performed on real and synthetic datasets shows that our approach outperforms a state-of-the-art in-DBMS solution supported by PostgreSQL by orders of magnitude.

Keywords cluster analysis, temporal-constrained (sub-)trajectory clustering, moving objects, indexing

[†] Nikos Pelekis
Dept. of Statistics & Insurance Science, University of Piraeus, Piraeus, Greece
E-mail: npelekis@unipi.gr

Panagiotis Tampakis Marios Vodas Yannis Theodoridis
Dept. of Informatics, University of Piraeus, Piraeus, Greece
E-mail: {ptampak, mvodas, ytheod}@unipi.gr

Christos Doulkeridis
Dept. of Digital Systems, University of Piraeus, Piraeus, Greece
E-mail: cdoulk@unipi.gr

1. Introduction

Nowadays, huge volumes of location data are available due to the rapid growth of positioning devices (GPS-enabled smartphones, on-board navigation systems in vehicles, vessels and planes, smart chips for animals, etc.). This explosion of data already contributes in what is called the Big Data era, raising new challenges for the mobility data management and exploration field (Giannotti et al. 2011; Pelekis and Theodoridis 2014).

Efficient and scalable trajectory cluster analysis is one of these challenges (Zheng 2015; Yuan et al. 2016). The research so far has focused on adapting well-known solutions that are effective for legacy data types to trajectory datasets. Thus, a typical approach is to transform trajectories to multi-dimensional (usually, point) data, in order for well-known clustering algorithms to be applicable. For instance, CenTR-I-FCM (Pelekis et al. 2011) builds upon a Fuzzy C-Means variant. Another approach is to focus on effective and efficient trajectory similarity search, which is the basic building block of every clustering approach. Once one has defined an effective similarity metric, she can adapt well-known algorithms to tackle the problem. For instance, TOPTICS (Nanni and Pedreschi 2006) adapts OPTICS (Ankerst 1999) to enable whole-trajectory clustering (i.e. clustering the entire trajectories), and TRACLUS (Lee et al. 2007) exploits on DBSCAN (Ester et al. 1996) to support sub-trajectory clustering.

Sub-trajectory clustering is a typical cluster analysis problem in Moving Object Databases (MOD). Fig. 1 illustrates a working example, i.e. a dataset consisting of four trajectories, T_1, …, T_4 (please note that the time dimension has been ignored for visualization reasons). Upon this dataset, the goal of sub-trajectory cluster analysis is to identify two clusters (coloured red and blue, respectively) and five outliers (coloured black). The challenging issue is to depart from the (entire) trajectory grouping into clusters, by first identifying the best partitioning of each trajectory into sub-trajectories and then performing cluster analysis upon those entities. For instance, due to the deviation of the trajectories illustrated in Fig. 1(a) at the end of their lifespan, clustering the entire trajectories might probably result to either no cluster at all (with all four trajectories labelled as outliers) or a single cluster consisting of all four trajectories; with the result depending on the sensitivity of the underlying trajectory similarity function and auxiliary parameters of the clustering algorithm. On the other hand, working at the sub-trajectory level we will be able to identify the red and blue clusters of sub-trajectories as well as the black outliers (see Fig. 1(b)).

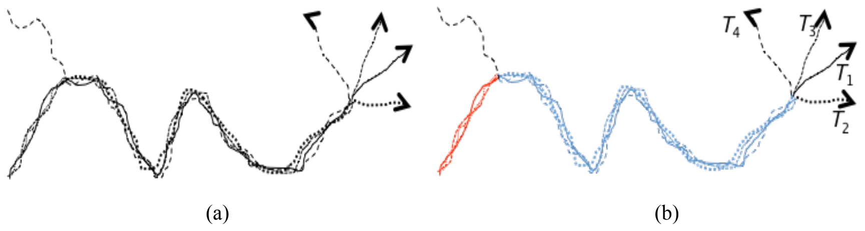


Fig. 1 Four trajectories (a) before (b) after sub-trajectory clustering

Finding a solution to the above described sub-trajectory clustering problem is challenging; what is even more challenging, is how one can support incremental and progressive cluster analysis in the context of dynamic applications, where (i) new trajectories arrive at frequent rates, and (ii) the analysis is performed over different portions of the dataset, and this might be repeated several times per analysis task.

As motivational example, consider the Location-based Services (LBS) scenario where LBS users transmit their trajectories to a central LBS server, e.g. when their trip is completed. From the server side, a MOD system is responsible for organizing user traces, aiming to support extensive (usually incremental and explorative) querying and mining processes. Since users (the data producers) transmit their location information in batch mode and asynchronously, the underlying data management framework should be able to handle this kind of information transmission. In other words, as we are especially interested in cluster analysis, the data server should be able to cluster users' trajectories in an incremental fashion. Clearly, the above techniques fail to meet such a specification.

Coming back to the example of Fig. 1, two main challenges need to be confronted: (i) given the addition of a new trajectory in the existing set of four trajectories, how can cluster analysis be performed over the updated data without applying the (quite expensive) clustering process from scratch, and (ii) how could we organize these trajectories so as to retrieve clusters valid in an ad-hoc temporal period of interest, without re-applying the clustering for the user-defined temporal period?

In this paper, we address the challenge of efficient and effective *temporal-constrained sub-trajectory cluster analysis*, by proposing an incremental and progressive solution to the problem. To this end, we propose a novel indexing scheme for large MODs, which is designed upon optimally selected samples of sub-trajectories, called *Representative Trajectories*, hence the term *ReTraTree*. Each sub-trajectory of this type acts as the representative of a group (cluster) of sub-trajectories. Thus, *ReTraTree* may be considered as a data structure that organizes (sub-)trajectories in a hierarchical fashion, while having small, but in any case adaptable, memory footprint. Based on its design, *ReTraTree* is able to incrementally partition and cluster trajectories as they are inserted in the MOD. Interestingly, the actual clustering process for the user-defined temporal period of interest, called *Query-based Trajectory Clustering* (*QuT-Clustering*), is performed as simply as a query execution upon the *ReTraTree*.

The contributions of our work are summarized below:

- we introduce the *temporal-constrained sub-trajectory cluster analysis* problem, which is a key problem for supporting progressive clustering analysis;
- we design *ReTraTree*, an efficient indexing scheme for large dynamic MODs, which is based on representative trajectories found in the dataset;
- as a solution to the problem of study, we devise *QuT-Clustering*, a sub-trajectory clustering algorithm running as simply as a query operator upon *ReTraTree*;
- we facilitate incremental trajectory cluster analysis by exploiting the incremental maintenance of *ReTraTree* along with the query-based clustering approach of *QuT-Clustering*;
- we perform an extensive experimental study upon real and synthetic datasets, which demonstrates that our in-DBMS implementation outperforms a state-of-the-art PostgreSQL extension by several orders of magnitude.

The rest of the paper is organized as follows: Section 2 formally defines the problem of *temporally-constrained sub-trajectory cluster analysis*. Section 3 presents the *ReTraTree* structure and its maintenance algorithms while Section 4 puts *ReTraTree* in action, in other words it provides the *QuT-Clustering* algorithm, also providing a complexity analysis of the entire framework. Section 5 presents our experimental study. Section 6 reviews related work. Section 7 concludes the paper and outlines future research directions.

2. Problem Setting

In this section, we provide the necessary definitions and terminology. Table 1 summarizes the definitions of the symbols used in the paper.

Table 1: Symbol table

Symbols	Definitions
D	a dataset $D = \{T_1, \dots, T_N\}$ of N trajectories
T	a trajectory of D, whose length is $\lvert T\rvert$ (in terms of number of points composing it)
$x_{t.s}$ ($x_{t.e}$)	starting (ending) timestamp of the time-varying object x, e.g. $T_{t.s}$ ($T_{t.e}$) is the minimum (maximum) timestamp of trajectory T
l	lifespan of D, namely the temporal period $[min(T_{t.s}), max(T'_{t.e}))$, $\forall$ $T, T' \in D$
p_i	i-th (3D) point of trajectory T, $p_i = (x_i, y_i, t_i)$
e_i	i-th (3D) line segment of T, $e_i = (p_i, p_{i+1})$
l_i	lifespan of line segment e_i, namely the temporal period $[t_i, t_{i+1})$
S	set of sub-trajectories partitioning trajectory T
S_i	i–th sub-trajectory of trajectory T
$V(e, e')$	voting function between two segments e and e' belonging to trajectories T and T', respectively
$V_T^{T'}$	voting descriptor of trajectory T with respect to T'
V_T^{D}	voting descriptor of trajectory T with respect to trajectory dataset D
V_D^{T}	voting of a trajectory dataset D with respect to T
$V_D^{D'}$	voting descriptor of trajectory dataset D with respect to trajectory dataset D'
R	Sample of representative sub-trajectories $R = \{R_1, \dots, R_M\}$
C	clustering of sub-trajectories in M clusters, $C = \{C_{R_1}, \dots, C_{R_M}\}$, $C_{R_i} \cap C_{R_j} = \varnothing$, i ≠ j, with sub-trajectory R_i representing cluster C_{R_i} of sub-trajectories
M	cardinality of C (and R)
Out	set of outlier sub-trajectories
W	the user-defined time window $(W \in l)$ for which we want to discover the sub-trajectory clusters

Definition 1 (*Voting between segments of two trajectories*): *Given two segments e and e' belonging to trajectories T and T', respectively, the voting function V(e, e') that calculates the voting e receives by e' is given by Eq.(1):*

$$V(e, e') = e^{-\frac{d^2(e,e')}{2\cdot\sigma^2}} \tag{2}$$

where the control parameter $\sigma > 0$ *shows how fast the function ("voting influence") decreases with distance.*

Since Euclidean distance $D(t)$ is symmetric, distance $d(e, e')$ is symmetric as well. As such, it holds that $V(e, e') = V(e', e)$; it also holds that $0 \leq V(e, e') \leq 1$. If the two segments are almost identical, i.e. distance $d(e, e')$ is close to zero, the voting function gets value close to 1. On the other hand, high values of distance $d(e, e')$ result in voting close to zero.

We can generalize the above discussion to define the *representativeness of a trajectory with respect to another trajectory*. Notice that the definition that follows is applicable to sub-trajectories as well (since a sub-trajectory is itself a trajectory, essentially a set of consecutive segments).

Definition 2 (*Voting descriptor and average voting of a trajectory with respect to another trajectory*): *Given a trajectory T of length |T| and another trajectory T', the voting descriptor* $V_T^{T'}$ *of T with respect to T' is a vector*

$$V_T^{T'}: \left(V(e_1,*), \ldots, V\left(e_{|T|-1},*\right)\right) \tag{3}$$

of dimensionality $|T|-1$ *where wildcard* $'*'$ *corresponds to the segment of T' that minimizes distance* $d(e_i, \cdot)$, $i = 1, \ldots, |T|-1$. *By* $avg\left(V_T^{T'}\right)$ *we denote the average of the values of the vector* $V_T^{T'}$ *of trajectory T with respect to trajectory T'.*

Obviously, the voting descriptor is not symmetric, i.e. $V_T^{T'} \neq V_{T'}^{T}$.

Definition 3 (*Voting descriptor of a trajectory with respect to a trajectory dataset*): *Given a trajectory dataset D and a trajectory T of cardinality |T|,* $T \notin D$, *the voting descriptor* V_T^D *of T with respect to D is a vector*

$$V_T^D: \left(\sum_{T' \in D} V(e_1,*), \ldots, \sum_{T' \in D} V\left(e_{|T|-1},*\right)\right) \tag{4}$$

of dimensionality $|T|-1$ *where wildcard* $*$ *corresponds to the segment of each T' in D that minimizes distance* $d(e_i, \cdot)$, $i = 1, \ldots, |T|-1$.

Recall that (i) the vote a segment can receive by another segment is a value ranging from 0 to 1, according to Eq.(2), and (ii) only one segment from each trajectory votes for a given segment of another trajectory, i.e. its nearest. This implies that the total voting – the sum of votes – received by a given segment is a value ranging from 0 (if all members of D vote 0) to N (if all members of D vote 1). To exemplify the above, back to the example of Fig. 1, voting descriptor $V_{T_1}^{\{T_2,T_3,T_4\}}$ presents in general higher values than voting descriptor $V_{T_4}^{\{T_1,T_2,T_3\}}$ since T_1 is more centrally located than T_4 in the dataset.

Definition 4 (*Voting of a trajectory dataset with respect to a trajectory*): *Given a trajectory dataset D of cardinality N and a trajectory T of cardinality |T|,* $T \notin D$, *voting* V_D^T *of D with respect to T is a value*

$$V_D^T = \sum_{T' \in D} avg(V_{T'}^{T}) \tag{5}$$

that accumulates the average voting of all trajectories $T' \in D$ *with respect to* T.

Definition 5 (*Voting of a trajectory dataset with respect to another trajectory dataset*): *Given a trajectory dataset D of cardinality N and another (reference) trajectory dataset D' of cardinality N',* $D \cap D' = \varnothing$, *voting* $V_D^{D'}$ *of D with respect to D' is a value calculated as follows:*

$$V_D^{D'} = \sum\nolimits_{T \in D'} V_D^T \tag{6}$$

Now we define the *temporally-constrained sub-trajectory clustering* problem that we address in this paper. Let W represent a time window within the lifespan of D, *i.e.* $W \in l$. *Further,* let D_W denote the set of sub-trajectories partitioning the trajectories in D, which are temporal-constrained within W. Formally:

Problem (*Temporal-constrained sub-trajectory clustering*): *Given (i) a trajectory database* $D = \{T_1, \ldots, T_N\}$ *of lifespan l, consisting of N trajectories of moving objects, and (ii) a time window W* ($W \in l$), *the temporal-constrained sub-trajectory clustering problem is to find: (a) a set* $C = \{C_{R_1}, \ldots, C_{R_M}\}$ *of M clusters of sub-trajectories,* $C_{R_i} \in D_W$, $i = 1, \ldots, M$, *around respective sub-trajectories* $R = \{R_1, \ldots, R_M\}$, $R_i \in C_{R_i}$, $i = 1, \ldots, M$, *called representative sub-trajectories, and (b) a set Out of outlier sub-trajectories,* $Out \in D_W$, *so that voting* $V_{D_W-R}^{R}$ *of dataset* D_W–R *with respect to R is maximized:*

$$(R, C, Out) = argmax\left(V_{D_W-R}^{R}\right) \tag{7}$$

The above problem is quite challenging, for a number of reasons. First, the segmentation (or partitioning) of trajectories found in *D* in sub-trajectories cannot be predefined nor is the result of a third-party trajectory segmentation algorithm, such as (Buchin et al. 2010; Lee et al. 2007; Li et al. 2010b). Instead, it is problem-driven: it is the clustering algorithm that solves the above problem that is responsible to find the best segmentation of trajectories into sub-trajectories. Practically, it is the clustering algorithm that is responsible to detect the red and blue parts of trajectories in Fig. 1, given that the analyst requires a clustering providing as time window *W* the whole lifespan of the dataset. Second, the optimization of the above scenario is a hard problem, since the solution space is huge. Third, one has to define the technique for selecting the set of the most representative sub-trajectories, whose cardinality *M* is unknown. Fourth, as already discussed, in a real MOD setting, the solution should support incremental updates. Put differently, data updates should be accommodated as soon as they come and update the existing clusters at low cost, instead of performing a new clustering process from scratch. Finally and most importantly, since clustering is applied over different portions of the dataset, and this might be repeated several times per analysis task, the solution to the problem should be repeatable for all the different time windows *W* that are of interest during explorative analysis. This comprises a novel feature and a major contribution of our work, since existing solutions for sub-trajectory clustering are not able to support progressive clustering analysis taking into account temporal constraints as filters.

3. The ReTraTree Indexing Scheme

We start this section with an overview of the *ReTraTree* indexing scheme (Section 3.1) and we continue with the algorithms that are necessary for its maintenance (Sections 3.2 – 3.4).

3.1 ReTraTree Overview

ReTraTree consists of four levels: the two upper levels operate on the temporal dimension while the 3rd level is built upon the spatio-temporal characteristics of the trajectories. The idea is to hierarchically partition the time domain by first segmenting trajectories into sub-trajectories according to fixed equi-sized disjoint temporal periods, called *chunks* (1st level partitioning). Then, each chunk is organized into *sub-chunks*, which form a partitioning of sub-trajectories within each chunk (2nd level partitioning). Notice that sub-chunks may overlap in time, i.e. they are not temporally disjoint.

Example 1 Fig. 2 illustrates six trajectories, $T_1,\dots, T_6$ spanning in two days (called Day 1 and Day 2). The dataset is split into two chunks at day-level, with mauve (green) colored sub-trajectories corresponding to the evolution of moving objects on Day 1 (Day 2, respectively). Furthermore, the chunk corresponding to Day 1 is subdivided to two sub-chunks, corresponding to $<T_1, T_2, T_3, T_4>$ and $<T_5, T_6>$, respectively. Although not illustrated in the figure, the first sub-chunk is valid during [20:00, 0:00) of Day 1 while the second sub-chunk is valid during [22:00, 0:00) of Day 1, thus they are overlapping in time. Especially for the first sub-chunk, we also illustrate the projection of the four trajectories on the spatial domain, which corresponds to Fig. 1(b).

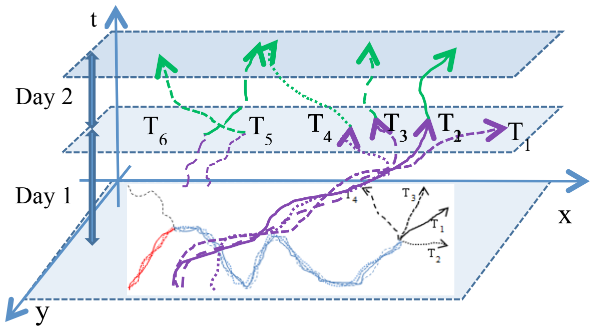


Fig. 2 Six trajectories, spanning in 2 days, split into daily chunks

Next, the sub-trajectories of each sub-chunk are clustered on the spatio-temporal domain with a sampling-based algorithm. In the previous example, this step results in the formation of two clusters of sub-trajectories (in red and blue) and five outlier sub-trajectories (in black), see Fig. 1(b). Thus, *ReTraTree* maintains only the representatives at the 3rd level of the structure, while the actual clustered data are archived at the 4th level.

Fig. 3 (and the paragraphs that follow) present *ReTraTree* in detail. Note that the top-three levels of the *ReTraTree* reside in main memory and only the 4th level is disk-resident.

1st level (chunks). The root of the *ReTraTree* consists of *p* entries, $p \geq 1$, corresponding to chunks sorted by time (in the example of Fig. 2, at daily level). Note that for each chunk H_i , $i = 1, \dots, p$, there is no need to maintain the actual temporal periods in the index nodes since they correspond to fixed equal-length splitting intervals. Each entry H_i maintains only a pointer to the respective set of sub-chunks $H_{i,n}$, $n \geq 1$, under this chunk. The set of all chunks forms the 1st level of the structure.

2nd level (sub-chunks). For each chunk, there is a set of sub-chunks, actually a sequence of triples < $H_{i,n}$.*per*, $H_{i,n}$.*R*, $H_{i,n}$.*Out* >, $n \geq 1$, where *per* is a temporal period [$per_{t.s}$, $per_{t.e}$) when the sub-chunk is valid (in the example of Fig. 2, [20:00, 0:00) and [22:00, 0:00), respectively for the two sub-chunks of Day 1), while *R* (*Out*) are pointers to the set of representative (outlier, respectively) sub-trajectories belonging to sub-chunk $H_{i,n}$. The sequence of triplets is ordered by <$per_{t.s}$, $per_{t.e}$>. The set of all sets of sub-chunks forms the 2nd level of the structure.

3rd level (cluster representatives). For each sub-chunk, the entries of set *R* consist of pairs <R_j, C_{R_j}>, $j \geq 0$, where each entry includes the representative sub-trajectory R_j and a pointer C_{R_j} to the subset of sub-trajectories belonging to that sub-chunk and forming a cluster around R_j. Note that $j = 0$ implies that there may exist sub-chunks with zero clusters (i.e. including outliers only). The set of all sets of cluster representatives (along with the pointers to actual data) forms the 3rd level of the structure.

4th level (raw trajectory data and outliers). The sets of actual sub-trajectories that compose clusters C_{R_j} are stored at the 4th level of the structure. For each sub-chunk $H_{i,n}$, there corresponds a set $D_{i,n}$ consisting of triples <*sub-trajectory-id*, C_{R_j}, *sub-trajectory-3D-polyline*> that keep the information about which sub-trajectory belongs to which cluster. On the other hand, set *Out* contains the outlier sub-trajectories of that sub-chunk. The outlier sub-trajectories are appropriately indexed in a 3D-R-tree structure (Theodoridis et al. 1996). The clustering process of sub-trajectories belonging to a sub-chunk, during which we detect sets *S* and *Out*, is a key process for *ReTraTree* and is described in detail in Section 5.

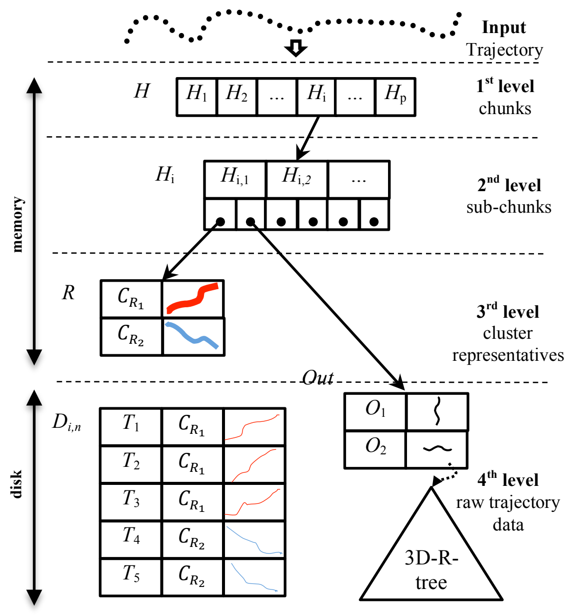


Fig. 3 Overview of the *ReTraTree* indexing scheme

How *ReTraTree* handles a new trajectory is discussed in the subsections that follow.

3.2 Hierarchical temporal partitioning

Given a trajectory database *D* of lifespan *l* (whose duration is denoted as |*l*|), a new trajectory *T*, and a fixed partitioning granularity *p*, applicable at the *ReTraTree* 1st level, *T* is partitioned into a number of sub-trajectories S_i, $i \geq 1$, where the sub-trajectory S_i is the restriction of *T* inside a temporal period p_i,

$$p_i = \left[\frac{|l| \cdot (i-1)}{p}, \frac{|l| \cdot i}{p}\right), 1 \leq i \leq p$$

where $^{|l|}/_{p}$ is the length of each time interval (i.e. the duration of the lifespan of each chunk) and timestamps $D_{t.s} + |l| * (i-1)/p, 2 \leq i \leq p$ are called *splitting timestamps*. As such, every trajectory in the dataset is partitioned into sub-trajectories using the same (pre-defined, according to granularity *p*) splitting timestamps. This chunking process is applied incrementally, whenever a batch of new recordings from a moving object arrives. In case of a new trajectory with temporal information that exceeds the last existing chunk, a new chunk is created and the set of chunks *CK* is extended.

At the 2nd level, each chunk is subdivided into (possibly, overlapping) sub-chunks. Specifically, a chunk is split into sub-chunks by grouping the sub-trajectories contained in the chunk, according to the following definition.

Definition 6 (*Grouping of sub-trajectories in the same sub-chunk*): *Given a temporal tolerance parameter τ and two sub-trajectories $S \in T$ and $S' \in T'$ belonging to the same chunk, these sub-trajectories can be grouped together in the same sub-chunk if their starting (ending) timepoints differ at most $\tau/2$, respectively. Formally, it should hold that:*

$$|S_{t.s} - S'_{t.s}| \leq {}^{\tau}/_{2} \wedge |S_{t.e} - S'_{t.e}| \leq {}^{\tau}/_{2} \tag{8}$$

Note that the above definition is not deterministic as there might be a sub-trajectory $S'' \in T''$ that also satisfies this condition. We handle this case by grouping the sub-trajectories when this condition is satisfied for the first time. Thus, we do not define and we do not search for a kind of "best-matching" sub-chunk. The reasons for this choice is that we are in favor of a very efficient insertion process, while we do not care about an optimal matching as this issue will be handled when the analyst asks for a clustering analysis. Regarding tolerance parameter τ, it is a user-defined parameter and can be exploited to impose an either stricter or looser notion of grouping. It also implies that e.g. when τ is set to 10 minutes, a sub-trajectory of less than 20 minutes duration cannot be grouped together with a sub-trajectory of more than 30 minutes duration.

3.3 Sampling-based sub-trajectory clustering

As already mentioned in Section 3, maximizing Eq. (7) is a hard problem. In order to tackle it we adopt a methodology for the optimal segmentation and selection of a sample of sub-trajectories from a trajectory dataset. Thus, in Fig. 4, we outline the *Sampling-based Sub-Trajectory Clustering* (*S^{2}T-Clustering*) algorithm, a two-step process that relies on a sub-trajectory sampling method, proposed in (Panagiotakis et al. 2012). Briefly, *S^{2}T-Clustering* relies on the output of the afore-mentioned sampling method (1st step), which is a set of sub-trajectories in the MOD that can be considered as representatives of the entire dataset. These samples serve as the seeds of the clusters, around which clusters are formed based on a greedy clustering algorithm (2nd step).

```
Algorithm S²T-Clustering
Input: MOD $D = \{T_1, T_2, \dots, T_N\}$, $\varepsilon$, $\delta$
Output: Sampling set $R$, Clustering $C$, Outlier set $Out$.
1.    $(R, S) \leftarrow$ Sampling($D$, $\varepsilon$)
2.    $(C, Out) \leftarrow$ GreedyClustering($R$, S, $\delta$)
3.    return ($R$, $C$, $Out$)
```

Fig. 4 *S^{2}T-Clustering* for building *ReTraTree* sub-chunks

The first step of *S^{2}T-Clustering* algorithm (line 1) invokes the *Sampling* method, which aims to solve an optimization problem, namely to maximize the number of sub-trajectories represented in a sampling set. In a few words, *Sampling* calculates the voting descriptor V_T^D of all trajectories T in D with respect to D, as described in Def. 3. Then, based on this signal, each trajectory is partitioned into sub-trajectories having homogeneous representativeness (i.e. the representativeness of all segments in a sub-trajectory does not deviate over a user-defined threshold), irrespectively of their shape complexity. According to (Panagiotakis et al. 2012), a trajectory should have at least w points in order for the segmentation to take place. Thus, w is an application-based parameter of *Sampling* that acts as a lower bound of the length of a trajectory under segmentation. Subsequently, *Sampling* selects a sampling set $R=\{R_1,\dots,R_M\}$ of sub-trajectories, which are hereafter considered as the representatives of D. Note that the number M of sub-trajectories is not user-defined; instead, it is dynamically calculated by the method itself. This is achieved by tuning *Sampling* with a parameter ε *($\varepsilon>0$ and $\varepsilon \rightarrow 0$)*, the role of which is to terminate the internal iterative optimization process when the optimization formula is lower than a given threshold (i.e. the ε parameter). Back to the example of Fig. 1, the above voting-and-segmentation phase would result in segmenting trajectory T_1 into three sub-trajectories (coloured red, blue, and black, respectively, in Fig. 1) according to its representativeness; similar for the rest trajectories of the MOD.) Then, *Sampling* would intuitively select two sub-trajectories as representatives, one from the blue sub-trajectories, and one from the red sub-trajectories.

At its second step (line 2), *S^{2}T-Clustering* uses sampling set R in order to cluster the sub-trajectories of the dataset according to the following idea: *each sub-trajectory in the sampling set is considered to be a cluster representative*. More specifically, clustering is performed by taking into account sampling set $R = \{R_1,\dots,R_M\}$ and vector of votes (i.e. representativeness) $V_{S_i}^{R_j}$ (actually we use the average voting $avg\left(V_{S_i}^{R_j}\right)$) between sub-trajectories of the original MOD $S_i \in D - R$ with respect to the representative sub-trajectories $R_j \in R$. Recall that $V_{S_i}^{R_j}$ (Def. 2) consists of $|S_i|$ elements, where each one represents the voting that the segments of S_i receive from the segments of R_j. To this end, in order for the *S^{2}T-Clustering* algorithm to maximize Eq. (7) for the special case where the time window W corresponds to the lifespan l of D, the cluster C_{R_j} of a representative sub-trajectory of the sampling dataset $R_j \in R$, i.e. the set of sub-trajectories that are assigned to cluster C_{R_j}, is provided by:

$$C_{R_j} = \left\{ S_i \in D - R : avg\left(V_{S_i}^{R_j}\right) \geq avg\left(V_{S_i}^{R_v}\right), \forall R_v \in R \wedge avg\left(V_{S_i}^{R_j}\right) \geq \delta \right\} \tag{9}$$

On the other hand, set *Out* of outliers consists of sub-trajectories that have been assigned to no cluster:

$$Out = \left\{ S_i \in D - R - C_{R_j}, \forall R_j \in R \right\} \tag{10}$$

The algorithm outlined in Fig. 4 simply iterates through all the representative sub-trajectories $R_j \in R$ of the sampling dataset R and applies the constraints of Eq. (9). Parameter δ is a positive real number between 0 and 1 that acts as a lower bound threshold of similarity between sub-trajectories and representatives. As such, it controls the size of the clusters C and the outlier set *Out*.

3.4 ReTraTree maintenance

S^2*T-Clustering* does not support arbitrary time windows nor dynamic data. The additional challenge that we have to address is to efficiently support such a clustering for arbitrary time windows and dynamic data. To achieve this, we need to efficiently support insertions of new trajectories in the *ReTraTree*.

The incremental maintenance of the *ReTraTree*, whenever a batch of recordings of a moving object (i.e. a trajectory T) arrives, is supported by the *ReTraTree-Insert* algorithm outlined in Fig. 5. We have already described how our method incrementally performs the first phase of partitioning in the time dimension (line 1). The *update_chunks* function returns the set of chunks H and the respective set of sub-trajectories S that correspond to the input trajectory T, i.e. the sub-trajectories S_i that intersect temporally with chunk H_i. Then, the algorithm assigns each sub-trajectory S_i to an appropriate sub-chunk (lines 2-4). This is actually checked by the *find_subchunk* function which, instead of applying Def. 6 between S_i and the other sub-trajectories in the sub-chunk, simply tests whether the following inequality holds: $|S_{i,t.s} - H_{i,n,t.s}| \leq \tau/2 \wedge |S_{i,t.e} - H_{i,n,t.e}| \leq \tau/2$. To gain this efficiency, the implicit assumption is that the temporal borders of each sub-chunk are left unchanged since its initialization with its first sub-trajectory. If there is not a matching sub-chunk with respect to time (line 5), a new sub-chunk is created, which is initialized with an empty representative set R, and an outliers set *Out* including the unmatched sub-trajectory (line 17). If there is an appropriate sub-chunk for the sub-trajectory under processing (line 5), the algorithm tries to greedily assign it to the best existing cluster (lines 6-13). If this attempt fails (line 14), the algorithm invokes *ReTraTree-Handle-Outlier* algorithm (outlined in Fig. 6).

```
Algorithm ReTraTree-Insert
Input: ReTraTree root, trajectory T, τ, ε, δ
1.  (H, S)←update_chunks(root, T)
2.  for each pair (H_i, S_i) ∈ (H, S) do
3.     clustered = false
4.     H_{i,n} = find_subchunk(S_i, H_i)
5.     if H_{i,n} ≠ ∅ then
6.        max_v_i = -1
7.        for each R_j ∈ H_{i,n}.R do
8.           if (non_common_lifespan(S_i, R_j) < τ) then
9.              v = $avg\left(V_{S_i}^{R_j}\right)$
10.          if (v ≥ δ AND v > max_v_i) then
11.             assign S_i to $C_{R_j}$
12.             max_v_i = v
13.             clustered = true
14.          if (clustered = false) then
15.             ReTraTree-Handle-Outlier(root, H_{i,n}, S_i, ε, δ)
16.    else
17.       update_chunk(S_i, H_i)
18. return
```

Fig. 5 ReTraTree-Insert algorithm

In particular, the second algorithm adds the sub-trajectory into the outliers' set of the sub-chunk, which acts as a temporary relation upon which *S*^{2}*T-Clustering* is applied, whenever the size of the relation exceeds a threshold α (e.g. α Mb that may correspond to a percentage of the dataset) with respect to its size, at the time of the previous invocation of the algorithm (line 2). Then, a new set of representative sub-trajectories will extend the existing set of representatives, only if it is δ-different from them (line 4). For each of the resulting new outliers, we re-insert the sub-trajectory from the top of the ReTraTree structure. This implies that we recursively apply *ReTraTree-Insert* for that sub-trajectory in order to search for other sub-chunks wherein it could be clustered or to form a new sub-chunk. This recursion is continued until an outlier is either clustered or partitioned to smaller pieces, due to successive applications of *S*^{2}*T-Clustering*. In case the size of an outlier becomes smaller than w, we archive it in the relation containing the raw data. Before applying a clustering analysis task and if the tree has been updated since the insertion of this specific trajectory, we give a last chance to these small outliers to be clustered by re-dropping them from the top of the structure. In other words, for a (sub-)trajectory T_k, if its length $|T_k| < w$ and T_k has not been assigned to a cluster, then, since it cannot be further segmented (and thus become again candidate to be clustered in a different sub-chunk); it cannot also be clustered before new trajectories update the tree.

```
Algorithm ReTraTree-Handle-Outlier
Input: ReTraTree root, sub-chunk $H_{i,n}$, outlier $S_i$, $\tau$, $\varepsilon$, $\delta$
1.  $H_{i,n}.Out \leftarrow H_{i,n}.Out \cup S_i$
2.  if $|H_{i,n}.Out| > \alpha$ then
3.      $(R, C, Out) \leftarrow S^2T\text{-}Clustering(H_{i,n}.Out, \varepsilon, \delta)$
4.      $H_{i,n}.R \leftarrow H_{i,n}.R \cup \{R' \subseteq R \mid \text{NOT } \delta\text{-}join(H_{i,n}.R, R)\}$
5.      for each outlier $O$ in $Out$ do
6.          if $|O| < w$ then
7.              archive $O$
8.          else
9.              ReTraTree-insert(root, $O$, $\tau$, $\varepsilon$, $\delta$)
10. return
```

Fig. 6 ReTraTree-Handle-Outlier algorithm

4. ReTraTree in Action

ReTraTree maintains clustered sub-trajectories at its leaves. However, given a temporal period, it is not enough to retrieve the clusters (i.e. the sub-trajectories "following" the representatives) that overlap this period. The reason is that the sub-trajectory clustering of overlapping sub-chunks may form representatives that: (a) are almost identical (as such, a *'merge'* operation should take place in order to report only one cluster as the union of the two (or more) clusters built around the similar representatives), and/or (b) can be continued by others (as such, an *'append'* operation should take place to identify the longest clusters, i.e. representatives).

In other words, an algorithm is required that takes *ReTraTree* as input and searches within it in order to identify the longest patterns with respect to the user requirements (e.g. discover all valid clusters during a specific period of time). This is made feasible through appropriate *'merge'* and *'append'* operations applied to the query results. To the best of our knowledge, such a query-based clustering approach is novel in the mobility data management and mining literature.

4.1 QuT-Clustering

Given two representatives R_i and R_j if (a) the two representatives have the same lifespan with respect to threshold τ and (b) the two representatives are also similar w.r.t. similarity threshold δ (this means that they origin from different sub-chunks), then this implies a *'merge'* operation. On the other hand, if (a) R_i ends close to the timepoint when the R_j starts with respect to threshold t, (b) the Euclidean distance of the last point of R_i is close (with respect to a distance threshold d) to the first point of R_j, and (c) a sufficient number of the same moving objects are represented by both representatives (with respect to a percentage threshold γ), this implies an *'append'* operation. Fig. 7 illustrates representatives of a chunk consisting of two chunks. A merge operation occurs between R_1 and R_2, whereas R_5 and R_6 will both be maintained in the final outcome although they have similar lifespans. An append operation occurs between R_3 and R_4.

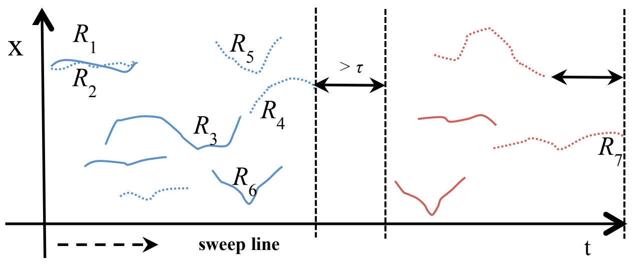


Fig. 7 Representatives of a chunk with two sub-chunks (dashed vs. continuous polylines) organized in a temporal priority queue of two groups (blue vs. red polylines)

Algorithm QuT-Clustering provided in Fig. 8 proposes such a solution on top of *ReTraTree*. The user gives as parameters the period of interest W, and the algorithm traverses the tree and returns clusters valid in this period.

```
Algorithm QuT-Clustering
Input: ReTraTree root, temporal period W
Output: Clusters C valid inside W
1. $H \leftarrow \{H_i, \mid overlap(root.H_i, W)\}$
2. for each $H_i \in H$ do
3.     $H_{i,n} \leftarrow \{H_{i,n}, \mid overlap(W, H_{i,n}.per)\}$
4.     $TEQ\_PQ \leftarrow bulk\_push\_TEQ(TEQ\_PQ, H_{i,n}, \tau)$
5.     while $TEQ\_PQ \neq \emptyset$ do
6.         $R \leftarrow temporal\_interleaving(R \cup TEQ\_PQ.pop())$
7.         for each $R_j \in R$ do
8.             $R_{overlap} \leftarrow temporal\_overlap(R_j, R, t)$
9.             for each $R_k \in R_{overlap}$ do
10.                if $(non\_common\_lifespan(R_j, R_k) < \tau)$ then
11.                    if $(avg\left(V_{R_j}^{R_k}\right) \geq \delta)$ then
12.                        $merge(R_j, R_k)$
13.                else if $(|R_{j,t.e} - R_{k,t.s}| < t)$ then
14.                    if $(euclidean\_dist(p(R_{j,t.e}), p(R_{k,t.s})) < d)$ AND $(common\_IDs(R_j, R_k) > \gamma)$ then
15.                        $append(R_j, R_k)$
16.                else
17.                    continue
18.    $R_{clustered} \leftarrow \{R_j \in R, \mid |R_{j,t.e} - H_{i,t.e}| > \tau\}$
19.    $R \leftarrow R - R_{clustered}$
20.    $C \leftarrow C \cup R_{clustered}$
21. return C
```

Fig. 8 ***QuT-Clustering*** **algorithm**

More specifically, the algorithm initially finds the chunks and then the sub-chunks that overlap the given period (lines 1-3). These sub-chunks are organized in a priority queue (line 4), which orders groups of representatives of the sub-chunks. Each group contains temporally successive representatives that are at most in temporal distance τ from each other. To exemplify this ordered grouping of sub-chunks, Fig. 7 shows the representative sub-trajectories (excluding outliers) of a single chunk, which consists of two sub-chunks, distinguished as dashed vs. continuous polylines. Note that for simplicity, y-dimension is omitted and specific borders of sub-chunks are not depicted, while the representatives form two groups, colored blue and red, respectively. Subsequently, the algorithm pops each group one-by-one and sorts all representatives with respect to time dimension, by interleaving the already sorted (from the step that constructs the priority queue) representatives coming from different sub-chunks (line 6). This is done by including representatives left from a previous round of the algorithm. Then, the algorithm sweeps the temporally interleaved representatives along the time dimension (line 7) and, for each of them, identifies the subset of its subsequent representatives in time that their lifespan overlap with the lifespan of the currently investigated representative, after the extension of the latter towards the future by t timepoints. For each pair of representatives R_j and R_k, the algorithm checks whether a merge operation (lines 10-12) or an append operation (lines 13-15) is necessary. In any other case (line 17) the algorithm simply continues with the next representative, and maintains both representatives intact. After each sweep, the algorithm maintains in the next round only those representatives that end at most τ seconds before the border of the current chunk (e.g. R_7, in Fig. 7), as candidates for merging with subsequent representatives (lines 18-20). The rest of the representatives are part of the final outcome of the algorithm.

Regarding the technical details, a *'merge'* operation practically maintains (in the working set of representatives R) one of the two representatives (e.g. the first) in the remaining process. The other representative is appropriately flagged so as to be able to retrieve the raw data that correspond to this cluster, if needed. For the *'append'* operation, we need to retrieve the identifiers of the sub-trajectories (not the sub-trajectories themselves) that correspond to the clusters implied by the representatives and apply a set intersection operation. This is facilitated by traditional indexing structures, such as by indexing the pair of representative id (i.e. cluster identifier) and sub-trajectory id of the raw data relation at the 4th level of *ReTraTree*. Practically, an *'append'* procedure replaces from the working set of representatives S the two representatives with one of those sub-trajectories that exist in both clusters. Note that the chosen sub-trajectory is selected randomly and it is the one used in the remaining process. Using another non-random choice at this step would be possible but not desired, as it would imply retrieval of the actual sub-trajectories. Finally, note that for simplicity reasons, we use the same threshold τ to compute the equivalence classes, as well as for considering whether two representatives refer to the same temporal period. In practice, these two easily configured parameters may be different, depending on the analysis scenarios pursued by the user. Similarly, threshold t corresponds to a small duration value, for instance, $t = 0$ in order to be as strict as possible.

4.2 Architectural aspects

The architecture of our framework is illustrated in Fig. 9. The core of the framework is the *ReTraTree* structure that is fed by either new incoming trajectories or data that have been processed in a previous round and could not be clustered. In both cases the *ReTraTree-Insert* algorithm handles the insertion. The trajectories are partitioned according to the in-memory part of the structure and stored on disk-based partitions. The trajectories assigned to an existing representative trajectory) are archived on disk in *clustered partitions*. Instead, trajectories that were not clustered are organized on disk in an (intermediate) *outlier partition*. When the size of the partitions exceeds a threshold, the *S²T-Clustering* algorithm applies the *Voting* process upon which the *Segmentation* of the trajectories takes place. The resulting sub-trajectories and their voting descriptors form the input of the *Sampling* module that selects new (i.e. non-existing) representatives that are back-propagated to the in-memory part of the *ReTraTree*. The new representative trajectories and the raw sub-trajectories form the input of the *GreedyClustering* module. If a sub-trajectory is clustered around a new representative, it is archived on disk. Otherwise it is an outlier and is re-inserted to *ReTraTree*, as it may now be accommodated in the index. This is due to its segmentation during the operation of *S²T-Clustering*, or due to the creation of new matching sub-chunks or representatives in the index. Finally, the analyst uses the *QuT-Clustering* algorithm to perform interactive clustering analysis by providing different time windows W as input.

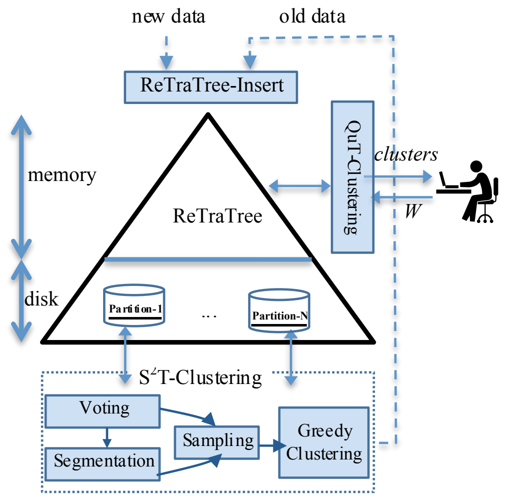


Fig. 9 ***Architectural aspects of ReTraTree***

4.3 Complexity Analysis

Concluding the discussion about our proposal, we provide a complexity analysis of (i) loading the *ReTraTree* structure and (ii) performing QuT-Clustering, according to the algorithms proposed so far. The assumption we make throughout our analysis is that the distribution of trajectories during the dataset's lifetime is uniform; in other words, selecting two random timepoints, t_i and t_j, the number of trajectories being 'alive' at t_i and t_j, respectively, remains more or less the same. In real world datasets, we do not expect to find perfect compliance to this, but we believe that this is a realistic assumption.

Lemma 1: *Under the uniformity assumption, the loading cost of the ReTraTree is:*

$$O\left(p \cdot (\overline{T_k} \cdot N + \bar{H} \cdot \bar{R}^2) \cdot log(\overline{T_k} \cdot N / \bar{H} \cdot \bar{R})\right)$$

where $\bar{H}$ is the average number of sub-chunks per chunk, $\bar{R}$ is the average number of representative sub-trajectories per sub-chunk and $\overline{T_k}$ denotes the average number of trajectory points in a database consisting of N trajectories.

Proof: Considering that $\bar{H}$ is the average number of sub-chunks per chunk and $\bar{R}$ denotes the average number of representative sub-trajectories per sub-chunk, *ReTraTree* can be considered as p balanced trees of $h = 2$ (excluding the root and the 3D-Rtrees found at the 1st and the 4th level of the structure, respectively) with the upper bound for the maximum number of leaves per tree being upper bounded by $\bar{H} \cdot \bar{R}$. Given the above, each sub-chunk has an average size of $\overline{T_k} \cdot N / \bar{H} \cdot \bar{R}$ segments. Setting threshold α of each sub-chunk to this value, *ReTraTree* will invoke the *S²T-Clustering* algorithm $O(p \cdot \bar{H} \cdot \bar{R})$ times [**result 1**].

Regarding the cost of *S²T-Clustering* algorithm, it is composed by the costs of its two components, namely *Sampling* and *Greedy-Clustering* (see Fig. 4). As it has been shown in (Panagiotakis et al. 2012), the most computationally intensive part of the *Sampling* method is the voting process with $O\left(\overline{T_k} \cdot N \cdot log(\overline{T_k} \cdot N)\right)$ cost for each trajectory in a database consisting of N trajectories indexed by a 3D-Rtree structure. Note that in our case, we maintain a forest of such trees, where each of them corresponds to the segments of the sub-trajectories that belong to the dynamically changing set of outliers of a sub-chunk. Therefore, in our case the number N of trajectories corresponds to the number of sub-trajectories that have been assigned to this set. Regarding *Greedy-Clustering*, as the voting vectors are pre-calculated during the *Sampling* step, its cost is dominated by the size of the representatives set $\bar{R}$. More specifically, the cost is $O\left(\bar{R} \cdot log(\overline{T_k} \cdot N)\right)$, i.e. the cost of performing $\bar{R}$ trajectory-based range queries in the database [**result 2**].

Since the size of the outliers of a sub-chunk set is estimated to be $\overline{T_k} \cdot N/\bar{H} \cdot \bar{R}$, the cost of the S^2T-*Clustering* algorithm in a sub-chunk is: $O\left(\overline{T_k} \cdot N/\bar{H} \cdot \bar{R} \cdot log(\overline{T_k} \cdot N/\bar{H} \cdot \bar{R}) + \bar{R} \cdot log(\overline{T_k} \cdot N/\bar{H} \cdot \bar{R})\right)$ [**result 3**].

By combining results 1-3 above (i.e. multiply the $p \cdot \bar{H} \cdot \bar{R}$ number of leaves with the cost of the S^2T-*Clustering* algorithm of a single sub-chunk), we have proven Lemma 1. ■

From a different point of view, the cost per trajectory insertion can be split in four parts: (i) the cost of chunking and sub-chunking the original trajectory to sub-trajectories, (ii) for each sub-trajectory, the cost of finding the matching representative, (iii) the cost of invoking the S^2T-*Clustering* algorithm, which is only in case that the sub-trajectory overflows threshold α of the sub-chunk, and (iv) the cost of checking whether the new representatives extracted by S^2T-*Clustering* can be inserted into the already identified representatives. Regarding the cost of each part, that of (i) is trivial, while that of (ii) and (iv) is $O(R)$ in both cases, since it implies a scan on the set of representatives, which however is small (R<<N). Obviously, the cost per trajectory insertion is dominated by the S^2T-*Clustering* algorithm.

Lemma 2: *Under the uniformity assumption, the cost of the QuT-Clustering algorithm is:*

$$O((\bar{H} \cdot \bar{R})^2)$$

where $\bar{H}$ is the average number of sub-chunks per chunk and $\bar{R}$ is the average number of representative sub-trajectories per sub-chunk.

Proof: As already shown, under uniformity assumption, each chunk maintains $O(\bar{H} \cdot \bar{R})$ representatives. Thus, invoking *QuT-Clustering* will eventually scan a number of $O(\left\lceil |W|/p \right\rceil \cdot \bar{H} \cdot \bar{R})$ representatives, where $\left\lceil |W|/p \right\rceil$ is the number of the involved chunks. However, at any time, the algorithm maintains a priority queue of $O(\bar{H} \cdot \bar{R})$ representatives (worst case scenario). Note that sorting this priority queue costs $O(\bar{H} \cdot \bar{R})$ only, since the sets of representatives of the corresponding sub-chunks are already sorted, thus a merge-sort performs the required temporal interleaving. Given this, as the representatives reside in memory and there is no special organization at this level, for each of these representatives the algorithm will scan all the other representatives in the worst case, thus leading to $O((\bar{H} \cdot \bar{R})^2)$ cost. ■

Interestingly, the cost of the *QuT-Clustering* algorithm is independent to the size of the database, thus it is a highly efficient solution for *progressive temporally-constrained sub-trajectory clustering* analysis. This is validated in the experimental study that follows.

5. Experimental Study

In this section, we present our experimental study. *ReTraTree* and its algorithms were implemented in-DBMS in Hermes[2] MOD engine over PostgreSQL, by using the GiST extensibility interface provided by PostgreSQL. More specifically, the top three levels of *ReTraTree* that reside in memory were implemented as temporary tables, while the 4th level was stored in traditional tables, upon which the 3D-Rtrees were built. Although our proposal is generic, we chose to put extra effort to implement it on a real-world MOD management system rather than an ad hoc implementation, because of the initially placed goal to support progressive clustering analysis. We argue that this is an important step towards bridging the MOD management and mobility mining domains, as state-of-the-art frameworks (Giannotti et al. 2011) could make use of the efficiency and the advantage of our proposal to execute clustering analysis tasks via simple SQL. This way, our approach becomes practical and useful in real-world application scenarios, where concurrency and recovery issues are taken into consideration.

All the experiments were conducted on an Intel Xeon X5675 Processor 3.06GHz with 48GB Memory running on Debian Release 7.0 (wheezy) 64-bit. We used PostgreSQL 9.4 Server with the default configuration for the memory parameters (`shared_buffers`, `temp_buffers`, `work_mem`, etc.). The outline of our experimental study is as follows: in Section 5.1, we discuss the setting of various parameters. In Section 5.2 we present baseline solutions with which we compare our proposals. In Section 5.3 we describe the datasets that we used in this study. In Section 5.4, we apply a qualitative analysis to verify that our proposal operates as expected by using datasets with ground truth. In Section 5.5 we provide a sensitivity analysis with respect to various parameters. In Section 5.6 we continue the qualitative evaluation of our approach in real datasets with general-purpose clustering validation metrics. In Section 5.7, we evaluate the maintenance of *ReTraTree* in terms of loading performance and size. In Section 5.8, we measure the I/O performance of *ReTraTree* with respect to the *QuT-Clustering* algorithm, the performance of which is assessed in Section 5.9.

5.1 Parameter settings

Regarding parameter settings, as our approach makes use of the sampling methodology of (Panagiotakis et al. 2012), we followed the best practices presented in that work. More specifically, the value of parameter σ was set to 0.1% of the dataset diameter, while that of ε was set to 10^{-3}. We would like to note that we made several

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

experiments by modifying the values of these parameters and the differences in the results were negligible, thus in a way we re-validated our earlier experience in the current setting.

As far as it concerns the parameters that affect the construction of *ReTraTree*, their effect is rather straightforward. Here we report our findings, which have been experimentally validated. More specifically, the more we increase p, the more chunks we create and hence the more the partitions (i.e. relations in our implementation). As the number of these partitions increases, the size and the construction time of *ReTraTree* decreases as the structure holds the same amount of data, but in smaller relations (i.e. smaller indexes). Moreover, as by increasing p we have a smaller structure size, the runtime of *QuT-Clustering* will be smaller. Regarding the τ parameter, the smaller it is, the more the number of sub-chunks and hence the more relations; thus, we fall at the previous case. In addition, the smaller the similarity threshold δ, the more the sub-trajectories that are assigned to already existing clusters. This implies that fewer sub-trajectories will end up to the outliers' set and hence the *S^2T-Clustering* algorithm runs fewer times. This means that the lower the δ the lower the construction time of the *ReTraTree*. Finally, regarding the value of α that is the threshold of the size of the outliers' set above which the *S^2T-Clustering* algorithm is applied, the more we increase α, the fewer times the *S^2T-Clustering* will run and consequently the smaller the construction time of *ReTraTree*. In our experiments we fixed threshold α to 5% of the dataset size.

In the subsequent sections we report on the effect of the important parameter of the time window W, while in Section 5.5 we particularly study the effect on both the efficiency and the quality of *QuT-Clustering* when varying the values of the remaining parameters, whose effect is not trivial to foresee without experimentation.

5.2 Baseline solution

To the best of our knowledge the *ReTraTree* structure and the corresponding *QuT-Clustering* algorithm is a novel solution to the *temporally-constrained sub-trajectory cluster analysis* problem and there is no comparable technique. Furthermore, as already mentioned, the *S^2T-Clustering* algorithm has some unique characteristics that make it appropriate as part of our solution. The most important characteristic is that it provides a greedy solution to the problem for the degenerated case where the time window W is equal to the entire lifespan of the dataset. This is a key observation that we exploit in our approach by organizing our data in sub-chunks consisting of sub-trajectories having the same lifespan and applying *S^2T-Clustering* to them. In Section 5.3 we demonstrate that the state-of-the-art TRACLUS algorithm (Lee et al. 2007) that is utilized also by the TCMM framework (Li et al. 2010b) cannot identify the clusters in datasets including ground truth. Moreover, in (Panagiotakis et al. 2012) it is shown that an efficient solution for the sampling process that the *S^2T-Clustering* algorithm utilizes, it requires a 3D-Rtree index.

Given the above, in this empirical study we set the following comparable pairs: (i) we compare the *ReTraTree* structure with the 3D-Rtree structure. A secondary but important reason for this choice is that 3D-Rtree is the prevailing structure that state-of-the-art spatial DBMS vendors have chosen to support in their products (e.g. PostGIS, Oracle Spatial); (ii) we compare the *S^2T-Clustering* algorithm with *QuT-Clustering* algorithm for the degenerated case where the time window W is equal to the entire lifespan of the dataset. Of course, our approach is applicable in any user-defined time window W. Thus, in this case the comparable pair is on the one hand the *QuT-Clustering* and on the other hand again the *S^2T-Clustering* algorithm after having restricted the dataset to the selected time window W. This implies that an analyst should first apply a temporal range query to restrict the dataset inside W, then build a 3D-Rtree on the restricted dataset and afterwards run the *S^2T-Clustering* algorithm. This is the best choice to perform a progressive clustering analysis without the *ReTraTree* and this is how the analysts work currently.

5.3 Datasets

In this study we used two real datasets (IMIS and GeoLife) and one synthetic (called SMOD); **Table 2** presents their statistics[3].

IMIS - IMIS is a real dataset consisting of the trajectories of 2,181 ships sailing in the Eastern Mediterranean for one week; data are collected through the Automatic Identification System (AIS) through which ships are obliged to broadcast their position for maritime regulatory purposes.

GeoLife - the GeoLife dataset (Zheng at el. 2010) contains 18,668 trajectories of 178 users in a period of more than four years. This popular 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, hiking, cycling, entertainment, sightseeing and shopping.

SMOD - Synthetic MOD (SMOD) consists of 400 trajectories and is used solely for the ground truth verification. 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) illustrated in Fig. 10.

[3] The GeoLife dataset is publicly available. The other two datasets are publicly available at chorochronos.datastories.org repository under the names 'imis7days' and 'smod', respectively.

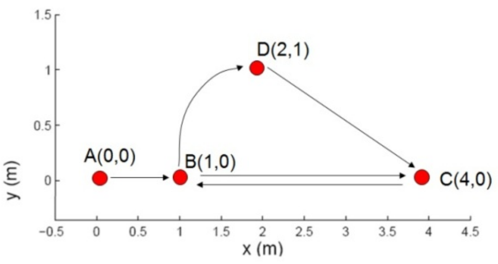


Figure 10. The 2-D map of SMOD with the three one-directional and one bidirectional road

We assume that half of the objects move with normal speed (i.e. 2 units per second) and the rest of them move with high speed (i.e. 5 units per second). To be more realistic, we have also added 50 db Gaussian white noise to the spatial coordinates of SMOD. The objects move under the following scenario (rules), for a lifetime of one hundred seconds: There exist three one-directional roads (A → B, B → D, D → C) and one bi-directional road (B ⇆ C). At t = 0 sec, all objects start from point A. Thus, the first destination of all objects is point B. Since half of the objects move with different speed, half of them (i.e. 200 objects) will arrive to point B at t = 20 sec and the rest of them at t = 50 sec.

When an object arrives at a destination point, 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 possible next point, it decides randomly about the next destination.

Table 2: Dataset Statistics

Statistic	SMOD	IMIS	GeoLife
# Trajectories	400	2,181	18,668
# Segments	35,273	12,516,337	24,159,325
Dataset duration	100 sec	7 days	~5 years
Avg. segment length (m)	8	169.5	72
Avg. segment speed (m/sec)	7.8	6	5
Avg. sampling rate (sec)	1	37	4
Avg. trajectory speed (m/sec)	2.9	3.7	3.9
Avg. # points per trajectory	89	5,739	1,295
Avg. trajectory duration	1.5 min	2.8 days	2.7 hours
Avg. trajectory length (km)	0.7	972.4	93

5.4 Quality of clustering analysis in synthetic datasets including ground truth

To the best of our knowledge there is no real trajectory dataset that provides ground truth that can be utilized for validating clustering techniques. Thus, our premise is to evaluate our approach qualitatively by using a synthetic dataset. The description of SMOD implies that the possible ending times of a moving object are t ≈ 20, t ≈ 50, t ≈ 80 or t ≈ 100. Based on this fact and by setting the chunk size equal to the duration of the dataset (i.e. 100 sec) we infer that the *ReTraTree* construction process should create 4 sub-chunks. We also infer the lifespan *l* of each sub-chunk. The invocation of the *ReTraTree-Insert* that builds these sub-chunks, concludes to apply the S^2*T-Clustering* algorithm in each of these sub-chunks, which in its turn results in discovering representatives (i.e. clusters) in each of them. This ground truth is illustrated in Table 3.

Table 3. The ground truth in SMOD

Sub-chunk	Path	Time periods (clusters)
$H_{1,1}$ $l = [0, 100]$	A→B	[0, 20], [0, 50]
	B→C	[20, 80], [50, 100]
	B→D	[20, 52], [50, 100]
	C→B	[80, 100]
	D→C	[52, 100]
$H_{1,2}$ $l = [0, 80]$	A→B	[0, 20], [20, 80]
	B→C	[20, 80]
$H_{1,3}$ $l = [0, 50]$	A→B	[0, 20] [0, 50]
	B→D	[20, 52]
$H_{1,4}$ $l = [0, 20]$	A→B	[0, 20]

For instance, sub-chunk $H_{1,1}$ with lifespan [0, 100] (i.e. objects that move through out the dataset's lifespan) includes eight representatives, for each of which we note its lifespan. For example, in $H_{1,1}$ there are two sub-trajectory clusters on the path A→B, with lifespans [0, 20], [0, 50], respectively.

We have loaded the SMOD dataset to the *ReTraTree*. We set the temporal tolerance parameter to $\tau = 2$ (i.e. we impose 1 second difference in the starting/ending timepoints). The resulting *ReTraTree* discovered indeed four sub-chunks with lifespans: [0, 100], [0, 81], [0, 54] and [0, 20]. By incrementally applying S^2*T-Clustering* in each of them, we resulted in the discovery of the representatives. Fig. 11 illustrates the representatives of the four sub-chunks. By combining each row in Table 3 with Fig. 10(a)-(d), we conclude that *ReTraTree* discovers the correct representatives, with their lifespans only slightly deviating from ground truth.

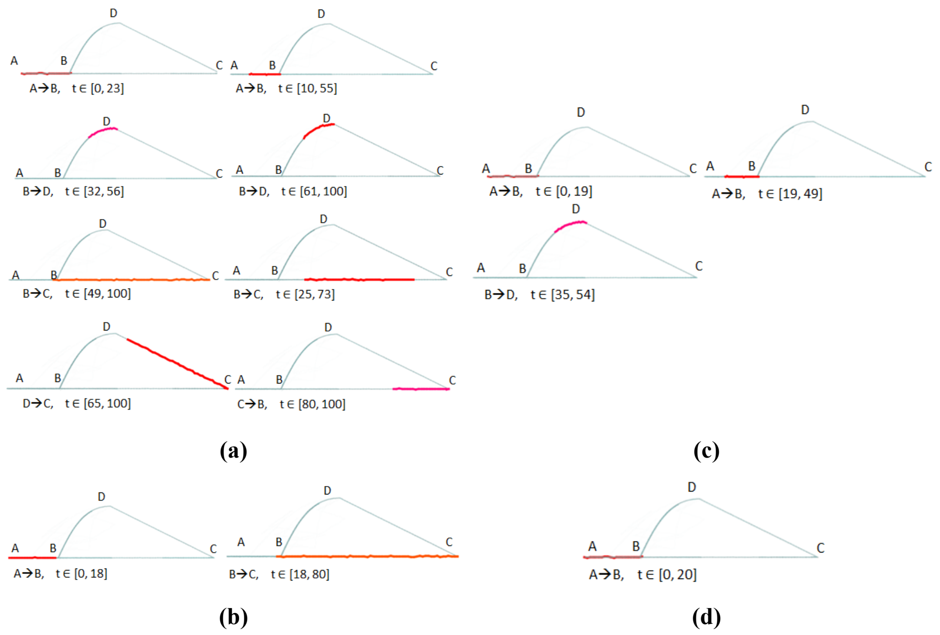


Fig. 11 The representatives of the four sub-chunks

We now investigate how the *QuT-Clustering* algorithm would operate by setting the temporal period *W* e.g. to the whole lifespan of the dataset. We used the values 5 sec, 10 m and 50% for (t, τ), d and γ respectively. After all append and merge operations take place, the resulting representatives are depicted in Fig.12, which is almost identical to the expected ground truth.

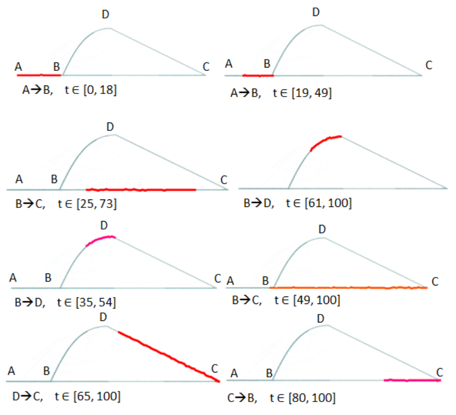


Figure 12: *QuT-Clustering* results with *W*=[0, 100]

In order to measure the stability of our method to noise effects, we have added more Gaussian white noise with Signal to Noise Ratio (SNR) level SNR = 30 db. The initial SMOD with additive noise of SNR = 50 db and the new SMOD with SNR = 30 db projected in 2-D spatial and 3-D spatiotemporal space is illustrated in Fig. 13. A small number of objects (i.e. outliers, four in our experiment) randomly move in space other than the roads that the other objects reside. These are also depicted in Fig. 13. In addition, the speed of outliers is updated randomly. Furthermore, for the sake of simplicity we assume that the chunk size is the whole lifespan of the dataset. According to this, the ground truth is restricted to the eight different paths that are valid for sub-chunk $H_{1,1}$.

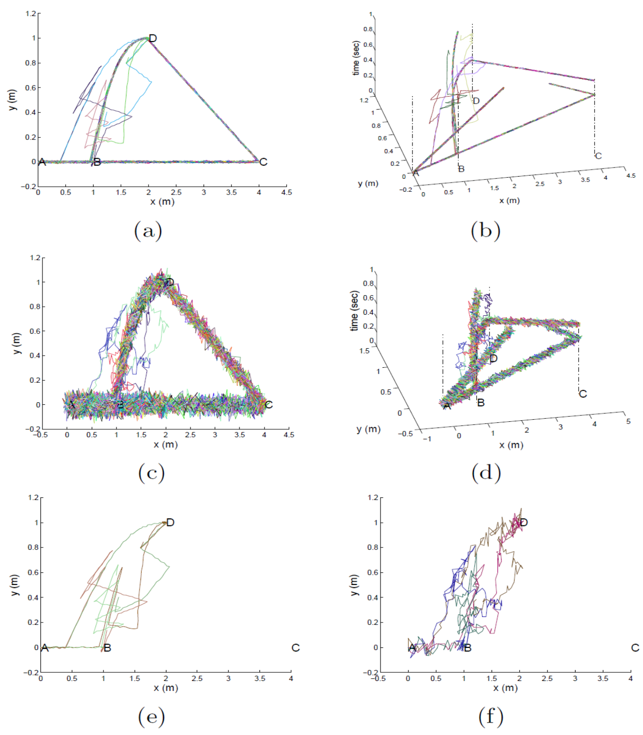


Figure 13: The trajectories of the SMOD with additive noise of SNR = 50 db projected in (a) 2-D spatial space ignoring time dimension and (b) spatiotemporal 3-D space. The trajectories of the SMOD with additive noise of SNR = 30 db projected in (c) 2-D spatial space and (d) spatiotemporal 3-D space. (e) The four outliers of the SMOD with additive noise of SNR = 50 db projected in 2-D spatial space ignoring time dimension. (f) The four outliers of our synthetic MOD with additive noise of SNR = 30 db projected in 2-D spatial space.

Given the above, and in order to demonstrate the benefits of *S^2T-Clustering* we compare with TRACLUS (Lee et al. 2007), the state-of-the-art sub-trajectory clustering technique. Again we assume that the chunk size is the whole lifespan of the dataset, hence the ground truth restricts to the eight different paths that are valid for sub-chunk $H_{1,1}$. In Fig. 14(a) and (b), we present the results of the *S^2T-Clustering* and TRACLUS, respectively. Specifically, in Fig. 14(a) we depict the selected sub-trajectories by *S^2T-Clustering* to serve as the pivots (i.e. representatives) for grouping other sub-trajectories around them, while in Fig. 14(b) we depict the synthesized representatives extracted (with *RTG* algorithm (Lee et al. 2007)) after the TRACLUS's grouping phase. Based on this experiment, it turns out that *S^2T-Clustering* effectively discovers all eight clusters (as well as the noisy sub-trajectories), thus *S^2T-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 (i.e. it discovers only four out of the eight clusters) due to the fact that it ignores the time dimension. Interestingly, note that TRACLUS discovers more or less linear patterns, ignoring the temporal information of the trajectories, as mentioned in (Lee et al. 2007).

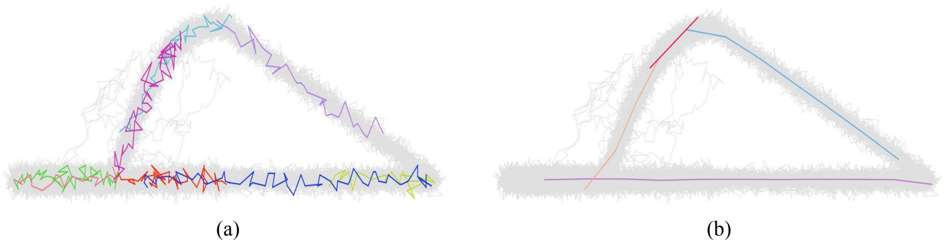


Figure 14: The representative trajectories (i.e. clusters) discovered by (a) *S^2T-Clustering* (b) TRACLUS

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 of sub-chunk $H_{1,1}$ 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^2T-Clustering* and calculated F-Measure; Fig. 15 illustrates this quality criterion by increasing the number of clusters. It is evident that *S^2T-Clustering* turns out to be very robust, achieving always precision and recall values over 92.3%, while the outliers are always detected correctly.

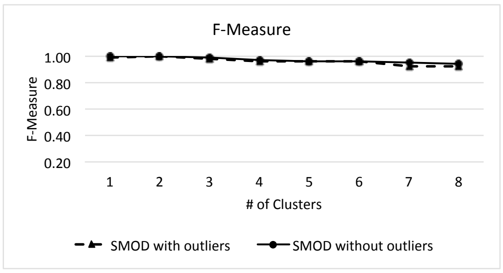


Figure 15. Quality of S^2T-Clustering w.r.t. number of clusters

5.5 Sensitivity analysis with respect to various parameters

In this section we first study the effect on the quality of the clustering result when varying the values of the parameters of the *QuT-Clustering* algorithm. Recall that *QuT-Clustering* does not change the clusters of the trajectories organized in *ReTraTree*. It returns modified representatives which are valid inside the given time window *W*, by merging or appending the initial representatives. Thus, the goal of the experiment is to measure the difference between the representatives resulted by the *QuT-Clustering* and the initial representatives. Intuitively, having this difference for different values of various parameters gives us a good hint about the sensitivity of *QuT-Clustering* w.r.t. the various parameters. To measure the difference, we employ the SSE metric between the initial representatives and their counterparts returned by *QuT-Clustering*. Obviously, if a representative is returned as-is, it contributes 0 to SSE. Apart from this set of experiments (one for each parameter), we further measure the execution time of *QuT-Clustering*, so as to study the effect of the parameters in the efficiency of the algorithm, in contradiction with its quality.

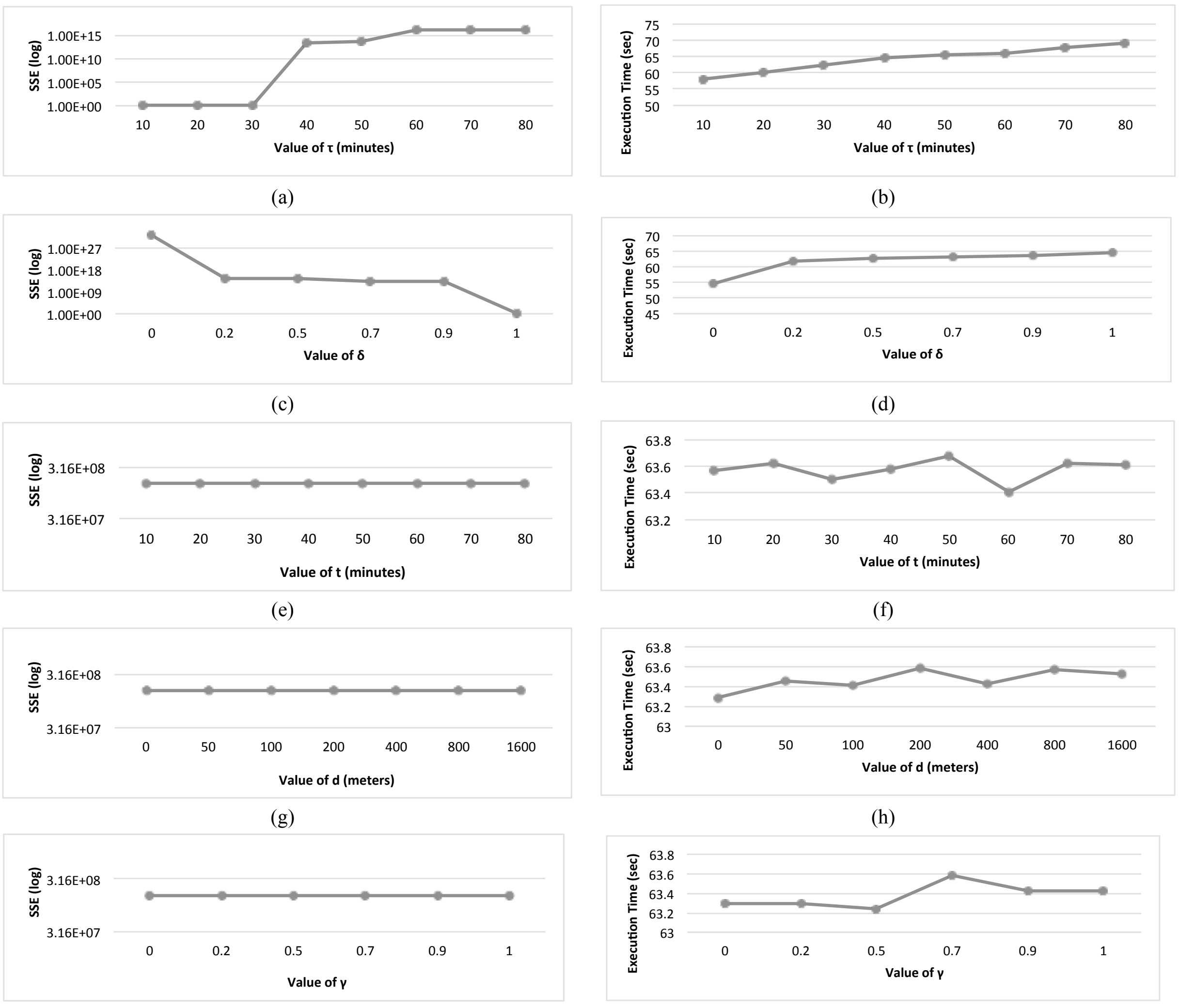


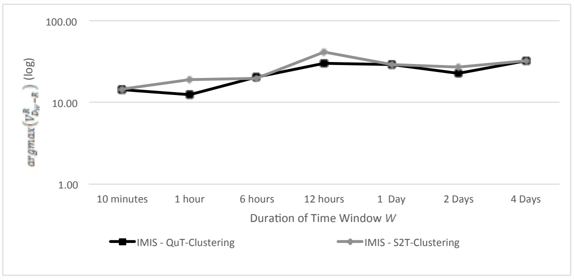
Fig. 16 (a)-(c)-(e)-(g)-(i) Sum of Square Errors, (b)-(d)-(f)-(h)-(j) Execution time, when varying the parameters of *QuT-Clustering*

The results of these experiments on IMIS dataset are depicted in Fig. 16. More specifically, as depicted in Fig. 16(a) as τ increases the quality drops due to the fact that we have more merges, hence the resulted representatives are more different than the original. The increasing number of merges results in gradual increase of the execution time (Fig. 16(b)). Fig. 16(c) shows that as δ increases the quality increases as fewer merges take place. Someone would expect that this would decrease execution time, however this is not the case as the costly operation in the merge phase is the calculation of the similarity between the two representatives, which is something that has already taken place. What we actually observe is a slight increase in the execution time, which occurs because *QuT-Clustering* ends up processing more representatives. Fig. 16(e), (g), (i) depict that quality is not affected by the different values of t, d and γ, respectively. The same conclusion stands also for the execution time illustrated in Fig. 16(f), (h), (j), respectively. This is because an append (raised when satisfying thresholds on these parameters) does not change the representatives themselves, i.e. an append simply returns two representatives as one.

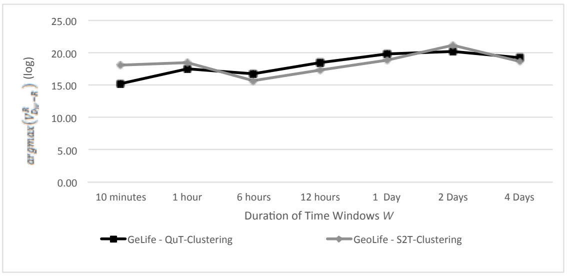
Given the above stable behavior of *QuT-Clustering* w.r.t. its parameters, in the rest of the experimental study the values of δ, d, γ, t and τ were set to the following intermediate values 0.7, 1km, 0.7, 30min and 30min, respectively.

5.6 Quality of clustering analysis in real datasets

In Section 5.4 we used a dataset including ground truth. In this section we use real datasets and general-purpose clustering validation metrics. Specifically, we evaluate the quality of the clustering through the $V_{D_{W-R}}^{R}$ measure introduced in Eq. (7). Note that this measure stands as an alternative to the Sum of Square Errors (SSE) and QMeasure used in the evaluation of TRACLUS (Lee et al. 2007), as it accumulates the (normalized) distances from the cluster centroids. More specifically, we use the IMIS and GeoLife real datasets and we compute $V_{D_{W-R}}^{R}$ for *S^2T-Clustering* and *QuT-Clustering* in different subsets of the two datasets. These subsets have been produced by selecting gradually coarser slices in the time domain. The time window W is set to lifespan of the subsets. The rationale of the experiment is that the $V_{D_{W-R}}^{R}$ of *QuT-Clustering* should be as close as possible to *S^2T-Clustering*, as the latter is a good solution of the problem for the degenerated case where the time window is equal to the lifespan of the dataset. Fig. 17 confirms that *QuT-Clustering* is able to identify clusters as well as *S^2T-Clustering* does. Put differently, *QuT-Clustering* results in representatives (after all the merge and append operations) which are very similar to those resulting from *S^2T-Clustering*, however as we will show in the subsequent sections, *QuT-Clustering* achieves this result with orders of magnitude better performance than *S^2T-Clustering*.



(a)

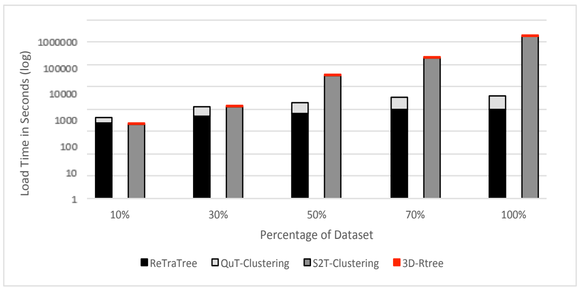


(b)

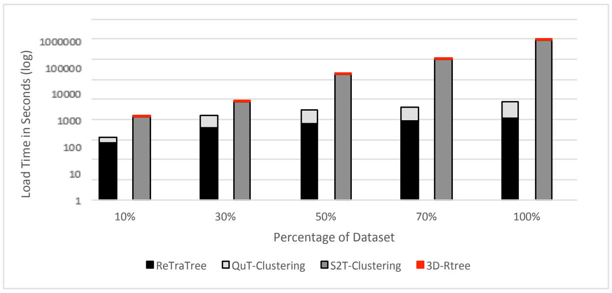
Fig. 17 $V_{D_{W-R}}^{R}$ of *QuT-Clustering* and *S^2T-Clustering* against batches of varying lifespan (setting *W* to their whole lifespan): (a) IMIS, (b) GeoLife

5.7 ReTraTree maintenance

In this section, we evaluate three different aspects of the *ReTraTree* structure, namely the efficiency of (i) loading and (ii) appending data, as well as (iii) the size of the structure. More specifically, the *Load* operation, measures the required time to load increasing volumes of data from scratch, which correspond to partitions of the MOD that are produced by selecting randomly a percentage of the total number of trajectories. Fig. 18 depicts the construction (loading) time to build, on the one hand, the *ReTraTree* and, on the other hand, the 3D-Rtree indices, i.e. the two alternatives to solve the problem at hand. Moreover, in order to correlate the required construction time of the indices with query time, we also add the execution time of the *QuT-Clustering* and the *S^2T-Clustering* algorithms, setting as time window W equal to the whole lifespan of each dataset. Note the log-scale on y-axis.



(a)

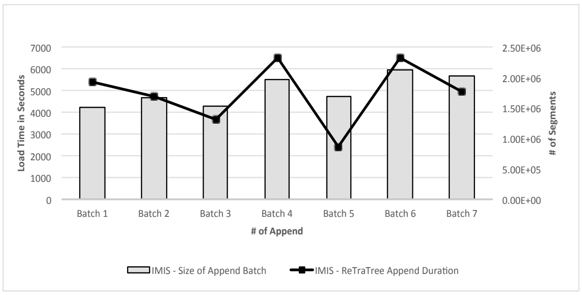


(b)

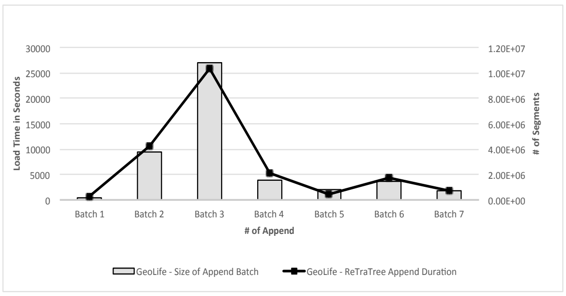
Fig. 18 Construction time of ReTraTree vs. 3DR-Tree (and execution time of *QuT-Clustering* and *S^2T-Clustering*) against datasets with increasing size: (a) IMIS, (b) GeoLife

From these results, we can make the following observations. First, the increase in loading time for *ReTraTree* is sublinear w.r.t. the dataset size, which is a positive testimony about its scalability. Second, when the total cost is considered (indexing and querying), it is clear that for large datasets our approach outperforms the competitor by two orders of magnitude. This is due to the fact that querying the *ReTraTree* is much more efficient than the 3D-Rtree, as the latter quickly becomes expensive, even for moderate dataset sizes. Put differently, *ReTraTree* harvests the increased construction cost in terms of fast query processing, thus boosting the performance of spatio-temporal clustering.

On the other hand, the *Append* operation measures the required time to append an existing *ReTraTree* with new batches of data, which correspond to new temporal periods – to perform this experiment we have split the datasets in 7 batches of equal duration (however, skewed size). Note that the first append operation loads data to an empty *ReTraTree*. Fig. 19 illustrates the time for each batch of data to be appended in the two index structures. Moreover, in the same figure we present the size of each batch. We observe that there is a very high correlation between batch size and batch execution time, perhaps with the exception of the first batch. This demonstrates that there exist no additional overheads as more batches are appended, thus the cost of Append mainly depends on the size of the appended batch. The fact that the first batch's execution time is disproportionate to its size has to do with the initialization cost of the *ReTraTree*.



(a)

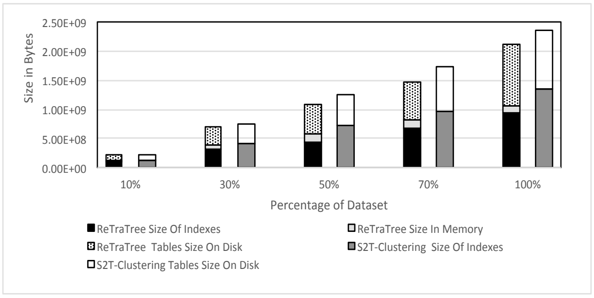


(b)

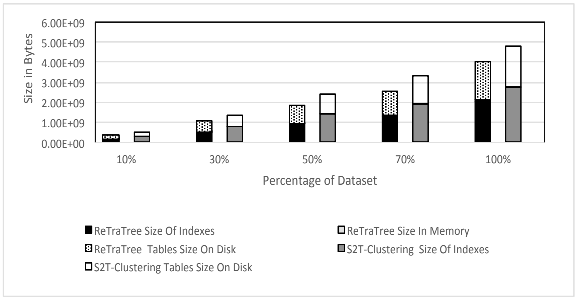
Fig. 19 Append of ReTraTree: (a) IMIS, (b) GeoLife

Next, we measure the size occupied by the structure. Fig. 20 depicts the size of the *ReTraTree* structure, both on disk and in memory, and compares it with the size occupied by the indices required for the *S^2T-Clustering* algorithm. As we have an in-DBMS implementation, the size of the indices is augmented with the required B-trees on the primary keys of the database tables.

For clarity, we also present the size of original tables, namely a single table for the *S^2T-Clustering* case and multiple tables for *ReTraTree*. As expected, we observe that the first three levels of *ReTraTree* have a small in-memory footprint, while, notably, our approach has a smaller size on disk in contrast to 3D-Rtree. This is due to that the *ReTraTree*'s partitioning scheme leads to more compact 3D-Rtrees (i.e. less dead space).



(a)



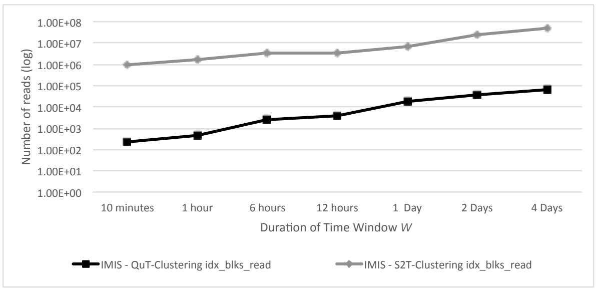
(b)

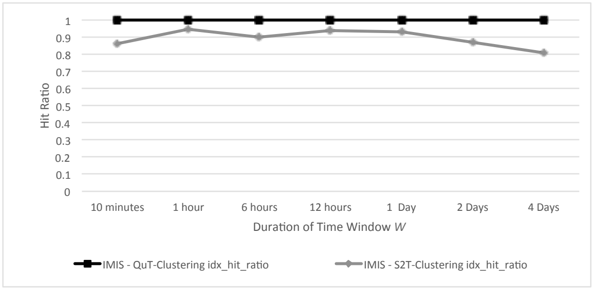
Fig. 20 Space requirements: (a) IMIS, (b) GeoLife

5.8 I/O performance

Now, we evaluate the I/O performance of both *QuT-Clustering* and S^2*T-Clustering* w.r.t. the number of index blocks read from disk (`idx_blk_read`) and the ratio of the index page hits (i.e. blocks read from cache) w.r.t. to all blocks (`idx_hit_ratio`).

Fig. 21(a) depicts the number of index blocks read from disk while increasing the duration of the time window *W* whereas Fig. 16(b) illustrates the hit ratio that clearly shows the advantageous use of the index in our case. The results are for IMIS dataset; we observed similar behaviour in GeoLife. Clearly, *QuT-Clustering* needs to access orders of magnitude fewer blocks to perform the clustering task, when compared to S^2*T-Clustering*. Moreover, this behaviour is consistent also for increased time windows.





(a) (b)

Fig. 21 *QuT-Clustering* vs. S^2*T-Clustering* (IMIS only): (a) blocks read from disk, (b) hit ratio

5.9 *Efficiency of QuT-Clustering* vs. S^2*T-Clustering*

As a final experiment, we measure the efficiency of performing the clustering task. The goal is to evaluate the retrieval of all the valid maximal clusters for varying time windows *W*, which is critical for progressive clustering analysis. We compare *QuT-Clustering*, with an approach that first extracts the relevant records using a temporal range query, then creates a 3D-Rtree index on the extracted values, and then applies S^2*T-clustering*. Fig. 22 depicts the execution time of both approaches (using a log scale on the y-axis) by varying the duration of the time windows *W*. Again, it is clear that for large datasets our approach outperforms the competitor by two orders of magnitude.

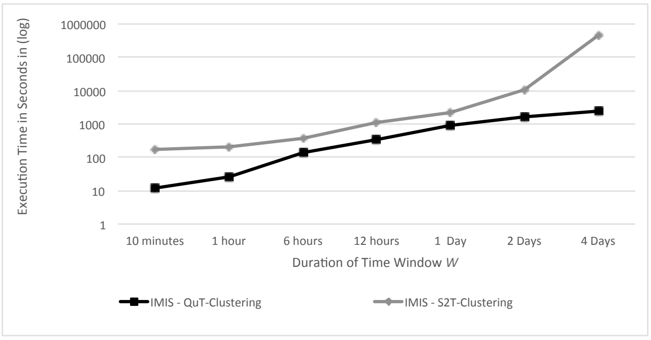


Fig. 22 Execution time of *QuT-Clustering* vs. S^2*T-clustering* by varying the datasets' lifespan (IMIS)

Moreover, we created a bundle of queries with random lifespan (i.e. time window *W*) and we executed them with random sequence. Fig. 23 depicts the accumulated execution time, i.e. time depicted for query *i*+1 also includes time required for query *i*. This experiment clearly shows a major benefit of *ReTraTree*. More specifically, S^2*T-Clustering* presents an excessive cost in performing multiple clustering tasks (with different time windows *W*), while in the case of *ReTraTree* this cost simply disappears. In *ReTraTree*, the overhead of performing a new clustering is negligible, as depicted by the almost straight line in the chart. Both results are for IMIS dataset; we observed similar behaviour in GeoLife (results omitted as they present no added value).

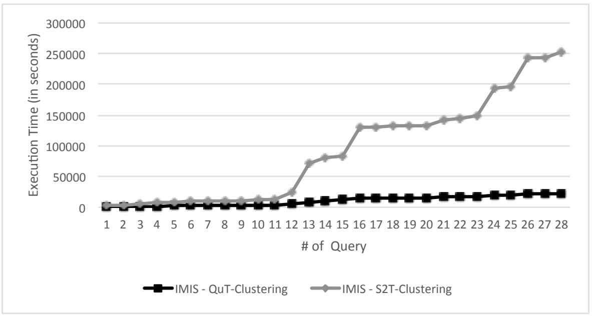


Fig. 23 Accumulated execution time of *QuT-Clustering* vs. S^2*T-clustering* w.r.t. a bundle of queries of random lifespan (IMIS)

6. Related Work

In recent years, the field of trajectory pattern mining has many success stories to narrate. These can be categorized according to the underlying mining methods used to discover the various behavioral patterns. Recent up-to-date surveys (Pelekis and Theodoridis 2014; Zheng 2015; Yuan et al. 2016) converge to a categorization of the various approaches that includes moving clusters, trajectory clustering, sequential patterns and periodic patterns. In this section, we review the state-of-the-art of the first two categories of patterns that directly relate to our work, while we also review the state-of-the-art indexing approaches.

6.1 Moving clusters

An interesting 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. One of the first approaches in this direction introduced the concept of flocks. A *flock* (Laube et al. 2005; Benkert et al. 2006) in a time interval I, where I spans for at least k successive timepoints, consists of at least m objects, such that for every timepoint in I, there is a disk of radius r that contains all m entities. If the objects change during the given interval, a kind of varying-flock is formed. Based on this idea, the notion of a *moving cluster* was introduced (Kalnis et al. 2005), which is a sequence of clusters c_1, …, c_k, such that for each timestamp i, c_i and c_{i+1} share a sufficient number of common objects. An extension of moving clusters pattern is the *convoy* pattern (Jeung et al. 2008) that is a group of objects that has at least m objects, which are density-connected with respect to a distance threshold e, during k consecutive timepoints. However, trajectories of real-world moving objects may meet together at some, nevertheless non-consecutive timepoints. To meet this real-world requirement, a *swarm* (Li et al. 2010a) is a collection of moving objects with cardinality at least m, that are part of the same cluster for at least k timepoints. It is important to note that the k timestamps are not required to be consecutive. The *traveling companion* (Tang et al. 2012) pattern is an approach for the online detection of convoy and swarm patterns from trajectories that arrive as a stream to the system. The *gathering* pattern (Zheng et al. 2013; Zheng et al. 2014) relaxes the constraints of the above-mentioned patterns by allowing the membership of a group to evolve gradually. Each cluster of a gathering should contain at least p participators, which are the objects appearing in at least c clusters of this gathering. The *gathering* pattern is used to detect events, thus, it requires that the region and its shape where the gathering takes place is more-or-less stable. Another approach that relaxes the globally consecutive timestamp constraint is the *platoon* pattern (Li et al. 2015), which only requires that the timestamps are locally consecutive. In other words, platoon patterns allow gap(s) in timestamps, but the consecutive time segments must have a minimum length.

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. None of the above approaches tackles the temporal-constrained sub-trajectory clustering problem. More concretely, it is out of their scope to organize the mined patterns in appropriate ways so as to report these patterns for arbitrary, user-defined time windows.

6.2 Clustering trajectory data

Most of the aforementioned approaches operate at the point level of the trajectories, meaning that they ignore the route between sampled points. Moreover, the sampled points are required to be at the same timestamps for all trajectories. Another line of research tries to discover groups of either entire or portions of trajectories considering the routes of the trajectories.

In the first of these categories, (Gaffney et al. 1999) proposed probabilistic algorithms for clustering entire trajectories using a regression mixture model. Subsequently, unsupervised learning is carried out by using EM algorithm to determine the cluster memberships in the model. Except from this probabilistic approach, researchers have followed two other directions. The first transforms the trajectories to a multi-dimensional space and then apply well-known clustering algorithms. This is because, the vast majority of the proposed clustering algorithms, such as k-means (MacQueen 1967), BIRCH (Zhang et al. 1996), CURE (Guha et al. 1998), DBSCAN (Ester et al. 1996), and OPTICS (Ankerst 1999), are tailored to work with point data, thus applying them to trajectory data is not possible. Unfortunately, it has been shown (Nanni and Pedreschi 2006) that such an approach based on k-means and hierarchical clustering algorithms leads to results of very poor quality. The second is to define appropriate similarity functions and embed them to extensible clustering algorithms. Following this line, there are several approaches whose goal is to group whole trajectories, including: T-OPTICS (Nanni and Pedreschi 2006) that incorporates a distance function (Frentzos et al. 2007) into the OPTICS algorithm (Ankerst 1999); the *vector field k-means* trajectory clustering technique (Ferreira et al. 2013) 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; CenTR-I-FCM (Pelekis et al. 2011) a variant of Fuzzy C-means; and the concept of *uncertain group pattern* introduced in (Wang et al. 2015). Both of the last two approaches propose specialized similarity functions having as goal to tackle the inherent uncertainty of trajectory data. Lately, another entire-trajectory clustering approach tackling uncertainty has been introduced in (Hung et al. 2015) where a 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). This 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. In (Xu et al. 2015) 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 address challenges in case of large variations within a cluster and ambiguities across clusters.

In the second category of sub-trajectory clustering (Lee et al. 2007) proposed TRACLUS, a partition-and-group framework for clustering 2D moving objects (i.e. TRACLUS ignores the time dimension) that enables the discovery of common sub-trajectories. TRACLUS clusters trajectories as line segments (sub-trajectories) by using an appropriate similarity function defined upon these directed line segments that is embedded in a variant of DBSCAN. The notion of the *representative trajectory* of a cluster is also defined. In this approach, the temporal information is not taken into consideration, while the partitioning is performed per trajectory having as criterion the line simplification of the trajectory. In (Panagiotakis et al. 2012) we have proposed a voting, a trajectory segmentation and a sampling algorithm that selects the top-k representative sub-trajectories of a trajectory database in order to support visual explorative analysis. In the current work, we make use of the sampling algorithm introduced in (Panagiotakis et al. 2012) as the first step of the *S^2T-Clustering* sub-trajectory clustering algorithm (see line 1, Fig. 4). The second step of *S^2T-Clustering* (see line 2, Fig. 4) is simply a greedy clustering algorithm that forms groups of sub-trajectories by assigning each sub-trajectory to its most similar representative sub-trajectory. Even though this is a simple idea, it results in a sub-trajectory algorithm that fulfils some unique requirements that the state-of-the-art TRACLUS algorithm cannot meet. This is clearly shown in our empirical study via a comparison of *S^2T-Clustering* with TRACLUS. However, this is not the core contribution of this work. We argue that the repetitive application of the *S^2T-Clustering*, in appropriately chunked portions of trajectories (according to *ReTraTree-Insert* algorithm, Fig. 5), properly organized in *ReTraTree*, is what we take advantage (in *QuT-Clustering algorithm*, Fig. 8) in order to efficiently solve the temporal-constrained sub-trajectory clustering problem.

The "Trajectory Clustering using Micro- and Macro- clustering" (TCMM) framework (Li et al. 2010b) is an incremental method that consists of two parts: (i) online micro-cluster maintenance and (ii) offline macro-cluster creation. The online part first simplifies trajectories by partitioning them into 2D line segments to find the spatial clusters of sub-trajectories; then, micro-clusters of the partitioned trajectories are computed and maintained incrementally. Micro-clusters hold and summarize similar trajectory partitions at very fine granularity levels. The offline part performs macro-clustering on the set of micro-clusters rather than on all trajectories when a user requests so. Our approach presents many crucial differences with respect to TCMM: (i) TCMM maintains and operates on summaries of trajectories (i.e. micro-clusters) only, while our approach maintains a lossless representation of the dataset in *ReTraTree*; (ii) TCMM applies spatial clustering on directed line segments (using (Lee et al. 2007)), while we operate on the native 3D space applying spatio-temporal clustering; (iii) the partitioning of the trajectories in TCMM is actually a simplification step taking place per trajectory, i.e. without global criteria, while in our proposal the partitioning is a cluster-aware task and it is an implicit outcome of the spatio-temporal clustering in the 3D space; (iv) most important, TCMM does not solve the temporal-constrained sub-trajectory cluster analysis problem as the offline macro-clustering part of TCMM targets at the entire lifespan of the database so as to identify global patterns without temporal constraints, while we are able to extract the clusters at any user given temporal period by simply querying the appropriately designed *ReTraTree* structure.

6.3 Indexing trajectory data

The ubiquity of R-trees in spatial databases has been expanded also in the domain of mobility data. To name but a few representative approaches, the 3D-Rtree for the purposes of spatiotemporal indexing was proposed in (Theodoridis et al 1996), while it was adapted to organize trajectories of moving objects in (Pfoser et al. 2000), where the TB-tree and STR-tree were introduced. The overhead introduced by representing trajectory segments as MBBs in a R-tree like structure was studied in (Hadjieleftheriou et al. 2006). MV3R-tree (Tao and Papadias 2001) is another efficient proposal for indexing the past movement of mobility data, consisting of a multi-version R-tree (extending the idea of multi-version B-tree) and a small auxiliary 3D-Rtree pointing to the leaf nodes of the former. A recent approach in trajectory indexing includes TrajStore (Cudre-Mauroux 2010), which is actually a storage scheme consisting of distinct spatial and temporal indexes. PA-tree (Ni and Ravishankar 2007) is a parametric index that organizes the coefficients of continuous polynomials approximating movement functions. All the above state-of-the-art indexing techniques make use of clustering methods so as to take advantage of their properties in the organization of the data (e.g. improve the compactness of the MBBs in R-tree-like structures).

From an abstract point of view, *ReTraTree* is a forest of 3D-Rtrees, however its design is generic, meaning that instead of using 3D-Rtrees in the leaves of *ReTraTree*, one could use any of the afore-mentioned specialized indices, most of which are variants of R-trees. We chose to use 3D-Rtrees as it is natively supported from most well-known spatial database systems. Most importantly, *ReTraTree* proposal is different, as we devise an indexing structure whose main goal is to efficiently support temporally-constrained sub-trajectory clustering analysis. To the best of our knowledge, this is novel in the domain of mobility data, as similar methods have only been introduced for legacy data types (Zhang et al. 1996).

7. Conclusions and Future Work

In this paper, we introduced the *temporally-constrained sub-trajectory cluster analysis* problem. To address it, we proposed *ReTraTree*, an indexing scheme which organizes trajectories by using an effective spatio-temporal partitioning technique. Partitions in *ReTraTree* correspond to groupings of sub-trajectories, which are incrementally maintained and represented via a hierarchical organization of a small (thus, light-weight in-memory) set of 'representative' sub-trajectories. Given this, the problem in hand can be efficiently solved as a query operator on *ReTraTree*, coined *QuT-Clustering*. Our approach further contributes to the mobility data management and mining domain for the additional reason that it has been designed and implemented in a MOD engine. Such functionality enables the application users to perform progressive cluster analysis via simple SQL in real extensible DBMS. Our extensive experimental study showed that our approach outperforms the state-of-the-art in-DBMS solution supported by PostgreSQL by several orders of magnitude.

Nowadays, mobility data is a clear representative of the 'Big Data' era. Although our proposal is orders of magnitude more efficient than state-of-the-art spatial DBMS, the execution time of a clustering analysis for a big dataset (even this is feasible) is not satisfactory. Thus, a limitation of our approach is that it is not directly applicable to big datasets, as it has not been designed having in mind an appropriate computational framework, e.g. the map-reduce paradigm. To this end, in the future, we plan to investigate real-time solutions, exploiting on modern in-memory big data architectures. Furthermore, our approach classifies sub-trajectories as outliers simply because they do not satisfy the conditions so as to be clustered around the discovered representatives. We leave as an interesting future direction of our work, to further study the semantics of the outliers, so as to be useful in specialized outlier detection applications.

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.

The work of P. Tampakis and C. Doulkeridis has been co-financed by ESF and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Aristeia II, Project: ROADRUNNER.

9. REFERENCES

Ankerst M, Breunig MM, Kriegel H-P, Sander J (1999) OPTICS: Ordering points to identify the clustering structure. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 49-60

Benkert M, Gudmundsson J, Hubner F Wolle T (2006). Reporting flock patterns. In *Proceedings of 14th Annual European Symposium (ESA)*, pages 660-671

Buchin M, Driemel A, van Kreveld M, Sacristán V (2010) An algorithmic framework for segmenting trajectories based on spatio-temporal criteria. In *Proceedings of of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 202-211

Cudre-Mauroux P, Wu E, Madden S (2010) Trajstore: An adaptive storage system for very large trajectory data sets. In *Proceedings of IEEE 26th International Conference on Data Engineering (ICDE)*

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 the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226-231

Ferreira N, Klosowski J.T, Scheidegger C.E and Silva C.T (2013) Vector Field k-Means: Clustering Trajectories by Fitting Multiple Vector Fields. In *Proceedings of EuroVis,* pages 201–210

Frentzos E, Gratsias K, and Theodoridis Y (2007) Index-based most similar trajectory search. In *Proceedings of IEEE 23rd International Conference on Data Engineering (ICDE)*

Gaffney S, and Smyth P (1999) Trajectory Clustering with Mixtures of Regression Models. In *Proceedings of the 5th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 63-72

Giannotti F, Nanni M, Pedreschi D, Pinelli F (2007) Trajectory Pattern Mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330-339

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 — The International Journal on Very Large Data Bases*, 20(5): 695-719

Guha S, Rastigi R, Shim K (1998) CURE: an efficient clustering algorithm for large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 73-84

Hadjieleftheriou M, Kollios G, Gunopulos D, Tsotras VJ (2006) Indexing Spatio-Temporal Archives, *The VLDB Journal — The International Journal on Very Large Data Bases*, 15(2): 143-164

Hung C.C, Peng W.C and Lee W.C (2015) Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *The VLDB Journal — The International Journal on Very Large Data Bases*, 24(2):169-192

Jeung H, Yiu ML, Zhou X, Jensen CS, Shen HT (2008) Discovery of Convoys in Trajectory Databases. In *Proceedings of the VLDB Endowment*, pages 1068-1080

Kalnis P, Mamoulis N, Bakiras S (2005) On discovering moving clusters in spatio-temporal data. In *Proceedings of the 9th international conference on Advances in Spatial and Temporal Databases (SSTD)*, pages 364-381

Laube P, Imfeld S and Weibel R (2005) Discovering relative motion patterns in groups of moving point objects. *International Journal of Geographical Information Science,* 19(6): 639–668

Lee JG, Han J and Whang KY (2007) Trajectory clustering: A partition-and-group framework. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 593-604

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

Li Z, Ding B, Han J and Kays R (2010) Swarm: mining relaxed temporal moving object clusters, In *Proceedings of the VLDB Endowment*, pages, 3(1-2): 723-734

Li Z, Lee JG, Li X, Han J (2010) Incremental clustering for trajectories, In *Proceedings of the 15th international conference on Database Systems for Advanced Applications (DASFAA),* pages 32-46

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281-297

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

Ni J and Ravishankar CV (2007) Indexing spatio-temporal trajectories with efficient polynomial approximations, *IEEE Transaction on Knowledge and Data Engineering*, 19(5): 663-678

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

Pelekis N and Theodoridis Y (2014) *Mobility Data Management and Exploration.* Springer.

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

Pfoser D, Jensen CS, Theodoridis Y (2000) Novel approaches to the indexing of moving object trajectories. In *Proceedings of the VLDB Endowment*, pages 395-406

Tang L.A, Zheng Y, Yuan J, Han J, Leung A, Peng W and Porta T.L (2013) A Framework of Traveling Companion Discovery on Trajectory Data Streams. *ACM Transactions on Intelligent Systems and Technology*, 5(1)

Tao Y and Papadias D (2001) MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. In *Proceedings of the VLDB Endowment*, pages 431-440

Theodoridis Y, Vazirgiannis M, Sellis T (1996) Spatio-temporal indexing for large multimedia applications. In *Proceedings of the 3rd IEEE International Conference on Multimedia Computing and Systems (ICMCS)*

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

Xu H, Zhou Y, Lin W and Zha H (2015) Unsupervised Trajectory Clustering via Adaptive Multi-Kernel-based Shrinkage. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)*

Yuan G, Sun P, Zhao J, Li D and Wang C (2016) A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, pages 1-22.

Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 103-114

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

Zheng Y, Xie X, Ma WY (2010) GeoLife: A collaborative social networking service among user, location and trajectory. In *IEEE Data Engineering Bulletin*. 33(2): 32-40

Zheng K, Zheng Y, Yuan N.J, Shang S (2013) On Discovery of Gathering Patterns from Trajectories. In *Proceedings of IEEE 23rd International Conference on Data Engineering (ICDE)*

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.